

版权注意事项：1、书籍版权归著者和出版社所有；
2、本PDF仅用于个人获取知识，进行私底下知识交流；
3、PDF获得者不得在互联网以任何目的进行传播；
如有需要，请尽量购买正版实体书！支持书籍作者！！

异步图书
www.epubit.com.cn

★ ★ ★ ★ ★
“十三五”


国家重点图书出版规划项目



Practical Machine Learning

实用机器学习

孙亮 黄倩 / 著

 中国工信出版集团

 人民邮电出版社
POSTS & TELECOM PRESS

孙亮



阿里巴巴数据科学与技术研究院高级专家。曾任微软Azure机器学习（Azure Machine Learning）部门高级数据科学家，先后毕业于南京大学计算机系（1999-2003）、中国科学院软件研究所（2003-2006）、美国亚利桑那州立大学计算机系（2006-2011），研究兴趣包括机器学习、数据挖掘及其实际应用等。近年来参加了KDD Cup、Heritage Health Prize等多项数据挖掘竞赛并多次取得优异成绩。在IEEE T-PAMI、NIPS、ICML、SIGKDD等机器学习领域的顶尖国际期刊和国际会议上发表论文近20篇，著有机器学习英文专著1部。

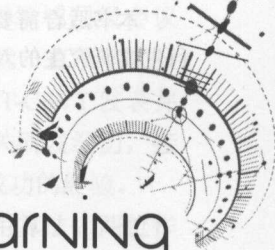
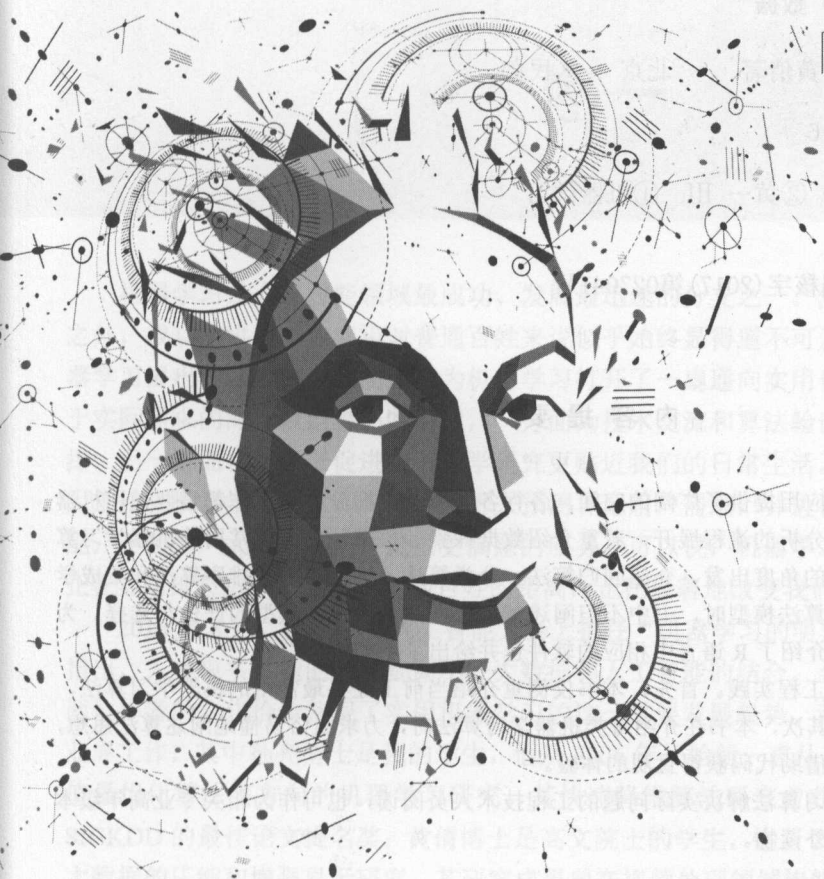
黄倩



河海大学副研究员，先后毕业于南京大学计算机系（1999-2003）、中国科学院计算技术研究所（2003-2010），研究兴趣包括多媒体大数据处理、机器学习、云计算等。参加过多个973、863、国家自然科学基金项目的研究，参与过AVS、H.265/HEVC等国内外视频压缩标准的制订。现主持包括国家自然科学基金在内的多个国家、省市级项目，并获南京市江宁区首批高层次创业人才“创聚工程”项目资助。在相关领域的知名国际期刊和国际会议上发表论文逾20篇，出版译著4本，参编专著1部。

★ ★ ★
“十三五”

国家重点图书出版规划项目



Practical Machine Learning

实用机器学习

孙亮 黄倩 / 著

人民邮电出版社
北京

图书在版编目 (C I P) 数据

实用机器学习 / 孙亮, 黄倩著. — 北京: 人民邮电出版社, 2017. 5
ISBN 978-7-115-44646-6

I. ①实… II. ①孙… ②黄… III. ①机器学习
IV. ①TP181

中国版本图书馆CIP数据核字(2017)第027041号

内 容 提 要

大数据时代为机器学习的应用提供了广阔的空间, 各行各业涉及数据分析的工作都需要使用机器学习算法。本书围绕实际数据分析的流程展开, 着重介绍数据探索、数据预处理和常用的机器学习算法模型。本书从解决实际问题的角度出发, 介绍回归算法、分类算法、推荐算法、排序算法和集成学习算法。在介绍每种机器学习算法模型时, 书中不但阐述基本原理, 而且讨论模型的评价与选择。为方便读者学习各种算法, 本书介绍了 R 语言中相应的软件包并给出了示例程序。

本书的最大特色就是贴近工程实践。首先, 本书仅侧重介绍当前工业界最常用的机器学习算法, 而不追求知识内容的覆盖面; 其次, 本书在介绍每类机器学习算法时, 力求通俗易懂地阐述算法思想, 而不追求理论的深度, 让读者借助代码获得直观的体验。

本书适合需要应用机器学习算法解决实际问题的工程技术人员阅读, 也可作为相关专业高年级本科生或研究生的入门教材或课外读物。

◆ 著 孙 亮 黄 倩

责任编辑 杨海玲

责任印制 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号

邮编 100164 电子邮件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

北京市艺辉印刷有限公司印刷

◆ 开本: 800×1000 1/16

印张: 22

字数: 490 千字

印数: 1—4 000 册

2017 年 5 月第 1 版

2017 年 5 月北京第 1 次印刷

定价: 79.00 元

读者服务热线: (010)81055410 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京东工商广字第 8052 号

序一

机器学习是人工智能领域最成功、发展最迅速的分支之一。然而多年来，除了人机对弈之外，成功的机器学习应用对普通百姓来说似乎始终显得遥不可及。大数据时代的来临对机器学习提出了新的挑战，同时也为机器学习打开了一扇通向实用化舞台的大门。近年来，源于实际需求的海量数据集不断出现，一方面为技术交流和算法验证提供了有力的基础数据支撑，另一方面也有助于促进相关科学研究更贴近我们的日常生活。例如，在滴滴，我们正致力于结合机器学习技术和大数据技术来预测各地用户需求，并进行实时的运力调度和订单分配，不断为广大司机和用户提供更满意的服务。可以说，机器学习技术和大数据技术的结合正在显著地拉近科研人员和普通百姓的距离，正在显著地改变我们的生产方式和生活方式。

2016年8月，我在中国人工智能大会上担任“机器学习的明天”专题论坛联席主席时曾指出，人类很多难题的解决都离不开大数据和人工智能的结合。今天，我欣喜地看到，本书两位作者的合作恰恰体现了实用机器学习的这一重要发展趋势。两位作者长期在学术界和工业界工作，其中孙亮博士是我的学生，他从2006年开始就一直从事降维算法、稀疏学习、数值最优化算法等方向的机器学习研究，其快速降维算法研究曾获得机器学习领域顶级会议SIGKDD的最佳论文提名奖；黄倩博士是高文院士的学生，他从2004年开始就一直从事视频大数据的压缩和增强显示研究，其研究成果曾在视频处理领域旗舰期刊IEEE T-CSVT发表并在工业界获得实际应用。两位作者博士毕业后均有在万人以上规模企业的相关研发经历，接触过各类实际问题和数据，对机器学习如何在实际中应用有着深刻的理解和成功的经验。

这本书不厚，但却覆盖了用机器学习技术解决实际问题的主要步骤和常用算法。两位作者从实际应用出发，介绍了数据探索、数据预处理、算法应用、性能评价等具体内容，并深入浅出地介绍了模型复杂度、损失函数等机器学习领域的基本概念。由于集成学习在实际中应用较为广泛，因此本书专列一章加以讨论。考虑到实践中大家更关注的是如何选择和使用算法，两位作者还使用R语言软件包来引导读者实际操作。与市面上对机器学习作一般性介绍的书籍相比，本书介绍的算法稍稍复杂一些，但也更加实用，书中讨论的内容正是实际应用机器学习解决问题时所需要掌握的内容。对于广大业界爱好者和相关专业研究生来说，这是一本理想的入门读物和参考书，因此我非常乐意向大家推荐本书。

叶杰平

滴滴研究院副院长，密歇根大学终身教授

2016年12月

序二

2016年10月3日，我从宿州匆匆赶回上海，试图从连日飨宴中恢复精力，等待5日凌晨谷歌公司的Pixel发布会。业界预测这将是一个划时代的发布会，标志着“Mobile First”向“AI First”的转型。这一场发布会，让我难掩激动地在朋友圈中留下了洋洋洒洒但未必成熟的千字技术点评。

我为什么会如此期待这场盛会，要从十多年前说起。那时我还是一个学生，学着和IT八竿子打不着的天文学。在本科结束后，响应高中读《时间简史》时内心深处的召唤，我继续学了天文学。经历短暂的欢愉之后，我终于意识到这条路的艰辛，我是不可能做这行了。我开始思考，有没有一条路能让我既接近最前沿的理论，又有足够好的实践环境呢？我在看了《AI》（《人工智能》）这部电影之后被震撼得天旋地转——人工智能可以创造出如此的美好，人工智能可以引发无穷的深思。

我开始去选修《机器人》的课程，开始去听张学工老师的《模式识别》课，开始把自己的知识往这个方向靠。所幸因为玻尔兹曼，这两个专业还是有些联系的。在一次选假期小班课程的时候，我选了一个SVM的讲座，后来当孙亮和我说他也听过这堂课的时候，我才意识到原来高人就在身边，只是我一直不曾留意。我曾看到过孙亮床位下一箱一箱的计算机书，大多与机器学习有关，只是人的神经系统是很奇妙的，当你不关注的时候，这些事情会自动在你的世界被屏蔽；当你关注的时候，这个概念会在极短的时间以各种方式在你面前展现。不久后我认识了同样以机器学习见长的黄倩，那时在周末大家都会找时间一起聚聚，当然话题大多是关于计算机的发展方向和算法的，然后我就作为旁听者被他们带进了门。

孙亮硕士毕业后去美国继续读机器学习的博士，最后到微软公司从事他擅长的机器学习和数据科学的研究；黄倩在高文院士指导下硕博毕业，最终回到高校任教，带领学子探索最前沿的算法。这些年他们一直躬耕一线，从未中断，我也在辗转中断断续续地做着数据相关的工作，最终转到Hadoop/Spark/TensorFlow开源平台。这几年大数据如火如荼，机器学习热浪汹涌，AlphaGo的成功进一步激励了业界，TensorFlow的开源、“AI First”的概念终于在坎坎坷坷10年之后开始盛行，高校的学子和一线的工程师开始被这个全新的世界吸引。同事和实习生不断地要我给他们推荐一些大数据和机器学习方面的书，我也曾给同事买过一些我看过的书。遗憾的是，这些书大多要么是纯理论的“屠龙术”，离实际应用还有一段距离，要么就是针对算法模块搞些例子，使学习者只知其然不知其所以然，还有的就是与目前业界普遍应用的算法不吻合。

对于一线的IT从业者和想要实践算法的学生来说，一本理论与实践相结合、能展现目前业界通用算法使用技巧的书，应该是大家最喜闻乐见的。这要求作者要有极深厚的理论功底，

能够把算法娓娓道来，还需要有足够的实践经验，娴熟业界各领域现在通用的算法。《实用机器学习》正是这样的一部佳作。本书详细讨论了机器学习中回归、分类、推荐、排序 4 类经典问题，详述了每一类问题中常用算法的理论来源，以及在 R 环境中如何去使用、评测和可视化展现。作为一线的实践者，书中对数据预处理也做了独立讲解，就理论过渡、全面性、实用性、易读性来说，本书都做了充分的考量。R 语言作为一个机器学习的平台工具，具有使用简单、分析方便、可用库完备以及可视化容易等特点，为进行问题分析和寻找原因提供了足够的便捷性。即使部分算法没有分布式的解决方案，本书讲授的实用机器学习算法，也极易在 Mahout 或 Spark MLlib 上平移对应的接口。在算法和实践平台上，本书倡导的方法和环境，几乎都可以和工业界做到无缝对接。

人工智能的世界来了，浩浩汤汤，开启了 IT 业者的另外一个世界。或许，我们只是需要一个开始，而《实用机器学习》来得适逢其时，你的一小步将来难说不是人类机器学习世界的一大步！

Fantasy (裴少芳)

威比网络科技(上海)有限公司大数据部总监

2016 年 12 月于上海

前 言

本书侧重于数据分析和机器学习的实践，涉及从原始数据搜集到建立模型解决问题再到算法性能评估的全过程。书中主要介绍实践中最常用的 4 类算法，包括回归算法、分类算法、推荐算法和排序算法。此外，书中还会介绍集成学习。集成学习是一类通过综合多个模型取长补短以取得更好效果的方法，对于回归、分类、推荐和排序问题都适用。在实践中，充分掌握这 4 类算法和集成学习即可解决相当多的实际问题。由于篇幅所限，聚类分析、关联规则等其他相关内容书中并没有一一介绍。

对于每种算法，本书首先介绍算法的原理。在理解算法原理和算法优缺点的基础上，读者在实践中就可以根据数据的特点和问题的具体需求选用合适的算法。为了突出算法的实践性，本书使用 R 语言中的软件包来介绍机器学习算法，特别是介绍了如何使用各种算法。R 语言是一种开源和免费的解释型语言，其最大的优点是提供了各种软件包，实现了各种不同的算法。机器学习中很多强大的算法在 R 中都有相应的程序包。我们在讲解各种机器学习算法时，都介绍了 R 中相应的软件包，并提供了相应的 R 程序来帮助读者学习这些软件包的使用。这样读者就可以通过 R 来直接使用相应的算法，获得数据分析的第一手建模经验。

除了介绍这 4 类机器学习算法之外，本书涵盖了使用机器学习解决实际问题的整个流程，包括数据探索、数据预处理、使用机器学习算法所构建的模型的评价和选择等。在实际使用机器学习处理数据的过程中，数据的探索和预处理是非常重要的步骤，在很多场合甚至比建立模型本身更加重要，从原始数据中提取出一个好的特征在很多时候能够显著地提高模型的性能。得到构建的模型后，我们还需要评价和选择模型。本书还会介绍不同类型算法对应的评价标准以及如何进行模型选择，并介绍 R 中的相关工具（如 caret 包），以帮助读者直接上手。

我们尽量使用简单通俗的语言来介绍机器学习中的基本概念和各种常用算法，并通过介绍 R 中对应的软件包来帮助读者迅速了解和掌握各种算法的使用。为了准确地介绍各类算法，不可避免地要用到一些数学知识，本书在第 3 章特别介绍了一些相关的数学知识。

本书的所有 R 代码（包括生成书中图的大部分 R 代码）都可以从人民邮电出版社异步社区（www.epubit.com.cn）网站上获得。

本书的出版得到了国家自然科学基金（61300122、61502145）的支持，得到了人民邮电出版社编辑杨海玲女士的支持和帮助，在此表示诚挚的谢意。成稿的关键时期适逢我们各自的女儿降生，在此衷心感谢双方家人的理解与支持。因水平和时间所限，书中难免有错误或不当之处，恳请广大读者不吝指正。读者若有任何问题或建议，可发送电子邮件至 sun.liang@outlook.com 或 huangqian@gmail.com。

孙 亮 黄 倩

2016 年 12 月分别于华盛顿雷德蒙和南京

目 录

第1章 引论1

1.1 什么是机器学习1

1.2 机器学习算法的分类2

1.3 实际应用3

1.3.1 病人住院时间预测3

1.3.2 信用分数估计4

1.3.3 Netflix 上的影片推荐4

1.3.4 酒店推荐5

1.3.5 讨论6

1.4 本书概述7

1.4.1 本书结构9

1.4.2 阅读材料及其他资源10

第2章 R 语言12

2.1 R 的简单介绍12

2.2 R 的初步体验13

2.3 基本语法14

2.3.1 语句14

2.3.2 函数17

2.4 常用数据结构19

2.4.1 向量19

2.4.2 因子23

2.4.3 矩阵24

2.4.4 数据框26

2.4.5 列表29

2.4.6 下标系统33

2.5 公式对象和 apply 函数34

2.6 R 软件包36

2.6.1 软件包的安装37

2.6.2 软件包的使用38

2.6.3 软件包的开发38

2.7 网络资源38

第3章 数学基础39

3.1 概率39

3.1.1 基本概念39

3.1.2 基本公式40

3.1.3 常用分布42

3.1.4 随机向量及其分布43

3.1.5 随机变量的数字特征46

3.1.6 随机向量的数字特征48

3.2 统计49

3.2.1 常用数据特征49

3.2.2 参数估计52

3.3 矩阵54

3.3.1 基本概念54

3.3.2 基本运算56

3.3.3 特征值与特征向量57

3.3.4 矩阵分解60

3.3.5 主成分分析62

3.3.6 R 中矩阵的计算68

第4章 数据探索和预处理74

4.1 数据类型74

4.2 数据探索75

4.2.1 常用统计量76

4.2.2 使用 R 实际探索数据76

4.3 数据预处理82

4.3.1 缺失值的处理82

4.3.2 数据的标准化83

4.3.3 删除已有变量85

4.3.4 数据的变换	86	6.2 决策树	130
4.3.5 构建新的变量: 哑变量	86	6.2.1 基本原理	130
4.3.6 离群数据的处理	88	6.2.2 决策树学习	131
4.4 数据可视化	89	6.2.3 过拟合和剪枝	138
4.4.1 直方图	89	6.2.4 实际使用	139
4.4.2 柱状图	92	6.2.5 讨论	148
4.4.3 茎叶图	95	6.3 逻辑回归	148
4.4.4 箱线图	96	6.3.1 sigmoid 函数的性质	148
4.4.5 散点图	100	6.3.2 通过极大似然估计来 估计参数	149
第 5 章 回归分析	104	6.3.3 牛顿法	151
5.1 回归分析的基本思想	104	6.3.4 正则化项的引入	153
5.2 线性回归和最小二乘法	105	6.3.5 实际使用	154
5.2.1 最小二乘法的几何解释	106	6.4 支持向量机	161
5.2.2 线性回归和极大似然估计	107	6.4.1 基本思想: 最大化分类间隔	161
5.3 岭回归和 Lasso	108	6.4.2 最大分类间隔的数学表示	163
5.3.1 岭回归	108	6.4.3 如何处理线性不可分的数据	164
5.3.2 Lasso 与稀疏解	110	6.4.4 Hinge 损失函数	166
5.3.3 Elastic Net	114	6.4.5 对偶问题	168
5.4 回归算法的评价和选取	114	6.4.6 非线性支持向量机和核技巧	170
5.4.1 均方差和均方根误差	114	6.4.7 实际使用	173
5.4.2 可决系数	114	6.5 损失函数和不同的分类算法	175
5.4.3 偏差-方差权衡	115	6.5.1 损失函数	175
5.5 案例分析	118	6.5.2 正则化项	178
5.5.1 数据导入和探索	118	6.6 交叉检验和 caret 包	180
5.5.2 数据预处理	120	6.6.1 模型选择和交叉检验	180
5.5.3 将数据集分成训练集和 测试集	121	6.6.2 在 R 中实现交叉检验 以及 caret 包	182
5.5.4 建立一个简单的线性回归 模型	121	6.7 分类算法的评价和比较	192
5.5.5 建立岭回归和 Lasso 模型	122	6.7.1 准确率	193
5.5.6 选取合适的模型	124	6.7.2 混淆矩阵	193
5.5.7 构造新的变量	126	6.7.3 精确率、召回率和 F1 度量	195
5.6 小结	126	6.7.4 ROC 曲线和 AUC	196
第 6 章 分类算法	127	6.7.5 R 中评价标准的计算	199
6.1 分类的基本思想	127	6.8 不平衡分类问题	201
		6.8.1 使用不同的算法评价标准	201
		6.8.2 样本权值	201

6.8.3 取样方法	202	8.4.2 IR-SVM 算法	266
6.8.4 代价敏感学习	203	8.4.3 RankNet 算法	267
第 7 章 推荐算法	205	8.4.4 LambdaRank 算法	271
7.1 推荐系统基础	205	8.4.5 LambdaMART 算法	273
7.1.1 常用符号	208	8.5 逐列方法	279
7.1.2 推荐算法的评价标准	209	8.5.1 SVM ^{map} 算法	279
7.2 基于内容的推荐算法	210	8.5.2 讨论	283
7.3 基于矩阵分解的算法	211	第 9 章 集成学习	284
7.3.1 无矩阵分解的基准方法	211	9.1 集成学习简介	284
7.3.2 基于奇异值分解的推荐算法	212	9.2 bagging 简介	285
7.3.3 基于 SVD 推荐算法的变体	216	9.3 随机森林	289
7.4 基于邻域的推荐算法	222	9.3.1 训练随机森林的基本流程	289
7.4.1 基于用户的邻域推荐算法	223	9.3.2 利用随机森林估计变量的 重要性	290
7.4.2 基于商品的邻域推荐算法	225	9.3.3 随机森林的实际使用	291
7.4.3 混合算法	226	9.4 boosting 简介	300
7.4.4 相似度的计算	227	9.4.1 boosting 和指数损失函数	301
7.5 R 中 recommenderlab 的实际 使用	232	9.4.2 AdaBoost 算法	302
7.6 推荐算法的评价和选取	250	9.4.3 AdaBoost 的实际使用	306
第 8 章 排序学习	253	9.4.4 讨论	311
8.1 排序学习简介	253	9.5 提升决策树和梯度提升算法	311
8.1.1 解决排序问题的基本思路	254	9.5.1 提升决策树和梯度提升算法的 基本原理	311
8.1.2 构造特征	255	9.5.2 如何避免过拟合	315
8.1.3 获取相关度分数	256	9.5.3 gbm 包的的实际使用	318
8.1.4 数学符号	257	9.5.4 讨论	327
8.2 排序算法的评价	257	9.6 学习器的聚合及 stacking	328
8.2.1 MAP	258	9.6.1 简单平均	328
8.2.2 DCG	260	9.6.2 加权平均	329
8.2.3 NDCG	261	9.6.3 stacking 的基本思想及应用	329
8.2.4 讨论	261	9.7 小结	331
8.3 逐点方法	262	参考文献	332
8.3.1 基于 SVM 的逐点排序方法	263	索引	334
8.3.2 逐点方法讨论	264		
8.4 逐对方法	265		
8.4.1 Ranking SVM 算法	265		

第1章 引论

随着计算机和互联网越来越深入到生活中的方方面面，人们搜集到的数据也呈指数级的增长。在这种情况下，大数据（big data）应运而生。大数据通常体量特别大，而且数据比较复杂，使得无法直接使用传统的数据库工具对其进行存储和管理。大数据带来了许多挑战，如数据的搜集、整理、存储、共享、分析和可视化等。广义的大数据处理涵盖了上述所有领域；狭义的大数据更多是指如何使用机器学习来分析大数据，从海量的数据中分析出有用的信息。

大数据分析的核心是机器学习算法。很多时候，我们有足够的数据，但是对如何利用这些数据缺乏理解。同时，实际问题往往比较复杂，并不能直接套用机器学习算法，我们需要对实际问题进行一些转化，使得机器学习算法可以应用。虽然实际问题表现形式各异，但是在将它们转化为机器学习能够处理的问题时，一般转化为如下4类问题：（1）回归问题；（2）分类问题；（3）推荐问题；（4）排序问题。这4类问题是实际应用中最主要的类型，覆盖了大部分实际问题。在1.3节，我们将详细介绍每类问题的具体例子。

1.1 什么是机器学习

机器学习（machine learning）是计算机科学的一个分支，也可以认为是模式识别（pattern recognition）、人工智能（artificial intelligence）、统计学（statistics）、数据挖掘（data mining）等多个学科的交叉学科。机器学习与数值优化（numerical optimization）也有很高的重合度。

机器学习研究如何从数据中学习出有效的模型，进而能对未来作出预测。例如，如果商店能够预测某一件商品在未来一段时间的销售量，就可以提前预订相应数量的商品，这样既可以避免缺货，又可以避免进太多货而造成积压。与传统的决策算法不同的是，机器学习算法依赖于数据。在前面的例子中，我们要从历史数据中学习出相应的模型以对未来进行预测。这样做有两个好处：第一，由于算法依赖于数据，可以使用新的数据来不停地更新模型，使得模型能够自适应地处理新的数据；第二，对人的介入要求少。在使用机器学习的过程中，虽然也会尽量利用人的经验，但更多地强调如何利用人的经验知识从数据中训练得到更好的模型。

目前，机器学习已成为研究和应用的热点之一。一些能够使用机器学习解决的实际问题包括：

- 根据信用卡交易的历史数据，判定哪些交易是欺诈交易；
- 从字母、数字或者汉字图像中有效地识别出相应的字符；

- 根据用户以往的购物历史来给用户推荐新的商品；
- 根据用户当前的查询和以往的消费历史向其推荐适合的网页、商品等；
- 根据汽车的发动机排量、年份、类型、重量等信息估计汽车的耗油量。

虽然这些问题的具体形式不同，但是均可转化成机器学习可以解答的问题形式。

从概念上讲，在机器学习中，我们的目标是从给定的数据集中学习出一个模型 f ，使得它能够有效地从输入数据中预测我们感兴趣的量。根据问题的不同，我们感兴趣的量（或者叫目标值）可以有不同的形式。例如，在分类问题中，目标值就是若干类别之一；在排序问题中，目标值就是关于文档的一个序列。

在机器学习中，通常我们解决问题的流程如下：

- (1) 搜集足够多的数据；
- (2) 通过分析问题本身或者分析数据，我们认为模型 f 是可以从数据中学习出来的；
- (3) 选择合适的模型和算法，从数据中学习出模型 f ；
- (4) 评价模型 f ，并将其利用在实际中处理新的数据。

在实际中，还需要根据应用的实际情况及时更新模型 f 。例如，若数据发生了显著变化，则需要更新模型 f 。因此，在实际部署机器学习模型时，上面的第3步和第4步是一个循环反复的过程。

一个经常与机器学习同时提起的相关领域是数据挖掘（data mining）。数据挖掘和机器学习在很多时候都被（不严格地）混用，因为这两者有很多重叠的地方。传统意义上，机器学习更加侧重于算法和理论方面，而数据挖掘更加注重实践方面。数据挖掘中的很多算法都来自于机器学习或者相关领域，少数来自于数据挖掘领域，如关联规则（association rule）。

另一个与机器学习关联很深的领域是统计学。在统计学中，我们学习了很多传统的处理数据的方法，包括数据统计量的计算、模型的参数估计、假设检验等。但在实际问题中，很多情况下我们并不能直接使用统计学中的方法来解决。一方面，随着数据规模的扩大，统计学中很多传统的数据分析方法需要通过大量的计算才能得到结果，时效性不高；另一方面，传统的统计学方法更多地考虑了算法在数学上的性质，而忽略了如何在实际中更好地应用这些算法。

1.2 机器学习算法的分类

在机器学习中，常用的算法可以分为监督型学习（supervised learning）和非监督型学习（unsupervised learning）^①。

- 在监督型学习中，除了输入数据 x 外，我们还知道对应的输出 y 。我们的目标是构建一个函数 $f(x)$ ，使得 $f(x)$ 能够预测输出 y 。

^① 在很多资料中还有第三类称为强化学习（reinforcement learning），近年来还有半监督型学习（semi-supervised learning）提出。本书主要涉及监督型学习和非监督型学习，不讨论强化学习和半监督型学习。

- 在非监督型学习中，我们只有输入数据 x ，没有对应的输出 y 。我们的目标是从数据中学习数据本身存在的模式 (pattern)。例如，聚类分析 (cluster analysis) 就是一个非监督型学习的典型例子，它通过分析样本之间的相似度来将样本划分为几个不同的聚类。

在监督型学习中，输出 y 一般称为目标变量 (target variable) 或者因变量 (dependent variable)，而输入 x 称为解释变量 (explanatory variable) 或者自变量 (independent variable)。

在实际中，在条件允许的情况下，我们偏好监督型学习。因为我们知道相应的目标变量的值，所以能够更加准确地构建模型，取得更好的效果。对于非监督型学习，在实际中，我们可以直接将其结果作为输出，但更多地是将其结果作为新的特征，再应用到监督型学习的算法中。例如，对于一组数据，可以先使用 k 均值算法对数据进行聚类分析，然后将聚类分析的结果作为新的特征。本书将主要讨论监督型学习。

在监督型学习中，一般将整个数据集分为训练集 (training set) 和测试集 (test set)。利用训练集中的数据，可以构建相应的模型 (model) 或者学习器 (learner)。利用测试集，可以估计所构建模型的性能高低。在数据集中，我们使用样本 (sample)、数据点 (data point) 或实例 (instance) 来称呼其中的每个点。监督型学习可以进一步分为回归问题、分类问题等。我们将在 1.3 节利用具体的例子来介绍监督型学习。

1.3 实际应用

在本节中，我们将会介绍一些可用机器学习解决的实际问题，包括病人住院时间预测、信用分数估计、Netflix 上的影片推荐和酒店推荐。每个例子都对应一类不同的机器学习问题。通过这些不同类型的机器学习问题，读者对机器学习可以有更多直观的感受。

1.3.1 病人住院时间预测

机器学习在医疗行业有着广泛的应用。我们以 Heritage Health Prize^① 竞赛作为例子以说明如何使用机器学习来预测病人未来的住院时间。

在美国每年都有超过 7000 万人次住院。根据相关统计，2006 年在护理病人住院上所花的无关费用就已经超过了 300 亿美元。如果我们能够根据病人的病历提前预测病人将来的住院时间，那么就可以根据病人的具体情况提前做好相关准备从而减少那些无谓的开销。同时，医院可以提前向病人发出预警，这样就能在降低医疗成本的同时提高服务质量。在从 2011 年开始的 Heritage Health Prize 竞赛 (HHP) 中，竞争者成功地使用机器学习的方法，由病人的历史记录预测了病人在未来一年的住院时间。图 1-1 显示了竞赛中使用的病历数据的一部分样本。

① <http://www.heritagehealthprize.com/c/hhp>

MemberID	ProviderID	Vendor	PCP	Year	Specialty	PlaceSvc	PayDelay	DSFS	PCG	CharlsonIndex	ProcedureGroup	SupL03
1 42286978	8013252	172193	37796	Y1	Surgery	Office	28	8- 9 months	NEUMENT	0	MED	0
2 97983248	3316066	726296	5300	Y3	Internal	Office	50	8- 9 months	NEUMENT	0	MED	1
3 2759427	2997752	140343	91972	Y3	Internal	Office	14	0- 1 month	METAB3	0	EM	1
4 73570559	7053364	240043	70119	Y3	Laboratory	Independent Lab	24	0- 1 month	METAB3	0	SCS	0
5 11837054	7557061	496247	68968	Y2	Surgery	Outpatient Hospital	27	4- 5 months	FXD1SLC	1-2	EM	1
6 45844561	1963488	4042	55823	Y3	Pediatrics	Office	25	3- 4 months	NEUMENT	0	EM	0

图 1-1 病历数据示例

1.3.2 信用分数估计

在现实生活中，向银行申请贷款是比较常见的，如房屋贷款、汽车贷款等。银行在办理个人贷款业务时，会根据申请人的经济情况来估计申请人的还款能力，并根据不同还款能力确定安全的借款金额和相应的条款（如不同的利率）。在美国，每个成年人都有相应的信用分数（credit score），用来衡量和评估借款者的还款能力和风险。

在估计申请者的还款能力时，需要搜集用户的多个方面的信息，包括：

- 收入情况；
- 年龄、性别；
- 职业；
- 家庭情况，如子女数量等；
- 还款历史，包括未按时还款的记录、还款金额等；
- 现有的各种贷款和欠款情况等。

如何将这些因素综合考虑从而决定借贷者的信用分数呢？直观地讲，可以使用一些简单的规则来确定信用分数。例如，某申请者的当前借款金额很高但收入一定，则进一步借款的风险很高，信用分数将会较低；又如，某申请者的某张信用卡在过去经常没有按时还款，则其信用分数也会较低。虽然使用简单的规则能够大致解决信用分数估计的问题，但是这个办法最大的问题是不能自适应地处理大量数据。随着时间的变化，申请者不还款的风险模型可能会发生变化，因此，相应的规则也需要修改。

银行通常可以得到海量的申请者数据和对应的历史数据。利用机器学习的方法，我们希望能从这些申请者过去的还款记录中自适应地学习出相应的模型，从而能够“智能”地计算申请者的信用分数以了解贷款的风险。具体地讲，在机器学习模型中，将申请者的信息作为输入，我们可以计算申请者在未来能够按时还款的概率。作为一个典型的例子，FICO 分数^①就是美国 FICO 公司利用机器学习模型开发出来的一个信用分数模型。

1.3.3 Netflix 上的影片推荐

Netflix 是美国的一家网络视频点播公司，成立于 1997 年，到 2015 年该公司已经有了近 7000 万的订阅者，并且在世界上超过 40 个国家或地区提供服务。Netflix 上的一项很重要的功能是根据用户的历史观看信息和喜好推荐相应的影片，如图 1-2 所示。2006 年 10 月至 2009

① <http://www.myfico.com/CreditEducation/WhatsInYourScore.aspx>

年9月, Netflix 公司举办了 Netflix Prize^① 比赛, 要求参赛者根据用户对于一些电影的评价 (1 星~5 星), 推测用户对另外一些没有看过电影的评价。如果能够准确地预测用户对于那些没有看过的电影的评价, 就可以相应地向这些用户推荐他们感兴趣的电影, 从而显著提高推荐系统的性能和 Netflix 公司的盈利水平。

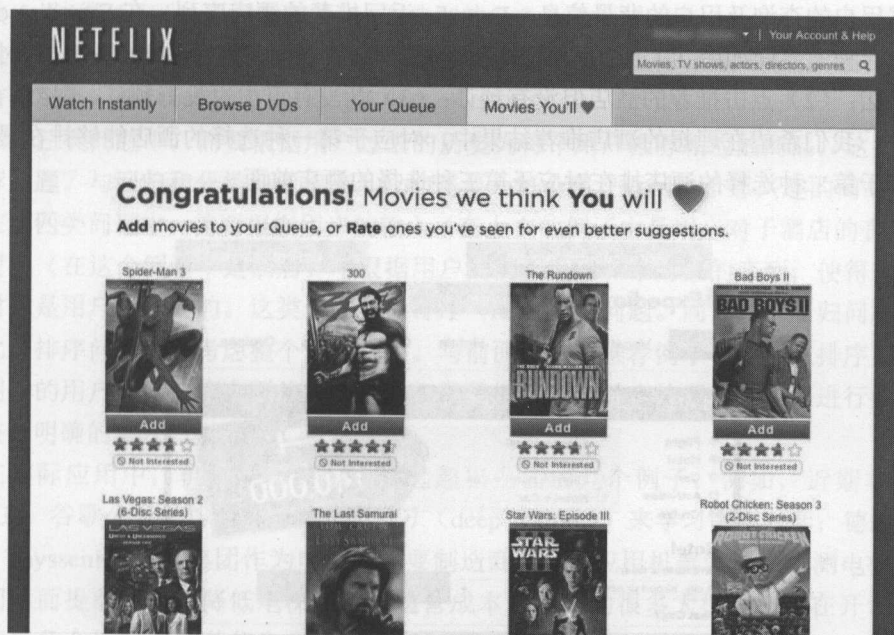


图 1-2 Netflix 上的电影推荐

在 Netflix Prize 比赛中, 获胜的标准是将 Netflix 现有推荐系统的性能提高 10%。在 2009 年, BellKor's Pragmatic Chaos 队赢得了比赛。其主要方法是基于矩阵分解的推荐算法, 并使用集成学习的方法综合了多种模型。Netflix Prize 比赛显著地推动了推荐算法的研究, 特别是基于矩阵分解的推荐算法的研究。在本书中, 我们也将详细介绍这些推荐算法。

1.3.4 酒店推荐

Expedia 是目前世界上最大的在线旅行代理 (online travel agency, OTA) 之一。它的一项很重要的业务是向用户提供酒店预订, 作为用户和大量酒店之间的桥梁。对于用户的每个查询, Expedia 需要根据用户的喜好, 提供最优的排序结果, 这样用户能够方便地从中选出最合适的酒店。

Expedia 于 2013 年年底与国际数据挖掘大会 (International Conference on Data Mining, ICDM) 联合举办了酒店推荐比赛。在该项比赛中, Expedia 提供了实际数据, 包括用户的查询以及其对所推荐结果点击或者购买的记录。在进行酒店推荐时, Expedia 考虑了如下因素:

^① <http://www.netflixprize.com/>

- 用户的位置和酒店的位置;
- 酒店的特征, 如酒店的价格、星级、位置吸引程度等;
- 用户过去预订酒店的历史, 包括价格、酒店类型、酒店星级;
- 其他竞争对手的信息。

根据用户的查询及用户的背景信息, Expedia 返回推荐的酒店序列。在 Expedia.com 上, 典型的酒店搜索界面如图 1-3 所示。根据返回的推荐结果, 用户有 3 种选择: (1) 付款预定推荐的酒店; (2) 点击推荐的酒店但没有预订; (3) 既没有点击也没有预订。显然, 根据用户的反应, 我们希望在理想的酒店推荐结果中, 对应于第一种选择的酒店能够排在最前面, 并且对应于第二种选择的酒店排在对应于第三种选择的酒店前面。

图 1-3 在 Expedia.com 上搜索酒店

1.3.5 讨论

上文中的 4 个例子分别对应于机器学习中的 4 类典型问题:

- 回归 (regression);
- 分类 (classification);
- 推荐 (recommendation);
- 排序 (ranking)。

在第一类问题中, 首先需要为每个病人构建一个特征向量 \mathbf{x} , 然后构建一个函数 f , 使得可以用 $f(\mathbf{x})$ 来预测病人的住院时间 y 。注意, 这里要预测的量 (病人的住院时间 y) 的范围是 $0 \sim$

365 (或者 366), 我们可以将其转化为回归问题。在回归问题中, 目标变量是一个连续值。

在第二类问题中, 需要为每个申请者构建一个特征向量 \mathbf{x} , 而输出 y 是 0 或者 1, 代表批准贷款或者不批准贷款。事实上, 输出 y 也可以是批准的概率。这是机器学习中典型的分类问题。在分类问题中, 目标变量 y 是一个离散变量。与回归问题类似, 我们的目标是构建一个函数 f , 使得 $f(\mathbf{x})$ 可以预测真实的 y 。在典型的两类分类 (binary classification) 问题中, 目标变量的取值为 0 或者 1 (有时是 -1 或者 1)。在多类分类 (multi-class classification) 问题中, 我们有多类, 而目标变量的取值是其中之一。

在第三类问题中, 需要根据用户过去的历史为每个用户推荐相应的商品, 这是一个典型的推荐问题。与回归和分类问题相比, 我们需要为每个用户返回一个感兴趣的商品序列。

在第四类问题中, 需要根据用户的输入 (在上文的例子中是用户对于酒店的查询), 从一系列对象 (在这个例子中是酒店) 中根据用户的需要返回一个对象的序列, 使得该序列最前面的对象是用户最想要的。这类问题称为排序 (ranking) 问题。同前面的回归问题和分类问题相比, 排序问题需要考虑整个返回序列。与前面的影片推荐例子相比, 在排序问题中我们需要明确的用户输入, 而在影片推荐中我们只是根据用户过去的历史信息来进行推荐, 用户没有进行明确的输入。

在实际应用中, 机器学习的应用远远超出上面的几个例子。例如, 近期非常热门的 AlphaGo, 谷歌公司在其中使用了深度学习 (deep learning) 来学习围棋对弈; 德国的蒂森克虏伯 (ThyssenKrupp) 集团作为电梯的主要制造商之一, 应用机器学习来预测电梯发生故障的时间从而提前维修, 降低电梯的综合运营成本; 美国的很多大型零售商在开设新店时, 都要搜集各个地区的各种信息和历史销售数据, 通过建立机器学习模型的形式选择最优的店址。

1.4 本书概述

本书主要从解决实际问题的角度来介绍常用的机器学习算法。在 1.3 节中我们讨论了机器学习中常见的 4 类典型问题, 基本上覆盖了目前实际中可以使用机器学习算法来解决的主要问题类型。在本书中, 我们将主要讨论对应的 4 类算法, 包括:

- 回归算法;
- 分类算法;
- 推荐算法;
- 排序算法。

其中回归算法和分类算法是两类最常用的算法, 也是其他很多算法的基础, 因此我们首先予以介绍。推荐系统在目前有了越来越多的应用, 而排序算法在搜索引擎等领域也获得了广泛的应用, 因此我们也会对常用的推荐算法和排序算法进行介绍。

在上面的 4 个例子中, 我们可以构建多个不同的模型, 希望它们之间能够取长补短, 使

得综合它们之后的模型的性能能够进一步提升。集成学习 (ensemble learning) 是一类通过综合多个模型以得到更好性能的方法, 对于回归问题、分类问题、推荐问题、排序问题都适用, 因此我们会专门用一章来介绍集成学习。

本书的目标是尽量介绍实用的算法。读者在掌握我们讨论的机器学习算法后就可以实际使用 R 中的软件包来解决实际问题了。对于每种算法, 我们首先介绍其基本原理。理解算法的基本原理非常重要, 它是我们实际使用算法的基础。只有理解了不同算法的特点, 才能在实际中根据不同数据的特点合理选择处理的算法。在本书中, 我们使用 R 语言来介绍这些算法的实际使用。机器学习中的各种常用算法在 R 中都有一种甚至多种实现, 而这些都是免费和开放的。用户可以直接使用 R 中对应的软件包来处理数据, 构建相应的机器学习模型。实际使用算法是学习机器学习算法最有效的方式之一。我们希望读者阅读完算法的理论部分后, 尽可能地使用 R 去实际处理数据和建立模型, 以获得用机器学习解决问题的第一手经验。

这里我们需要强调, 很多实用、有效的算法并不简单, 甚至比较复杂。以决策树为例, 其原理简单, 在很多关于机器学习的书籍中都是予以重点介绍的, 但是, 由于决策树对于噪声比较敏感, 在实际中很容易出现过拟合的现象, 因此很少直接使用。然而, 基于决策树的随机森林和提升树在实际中应用极为广泛。在本书中, 我们也会介绍决策树, 但主要目的是作为介绍随机森林和提升树的基础。

由于篇幅有限, 本书将主要集中介绍那些最实用的算法。例如, 我们没有介绍聚类分析 (cluster analysis) 和关联规则 (association rule)。此外, 我们也省略了一些常用算法。例如, 在分类算法中, 朴素贝叶斯分类器 (Naïve Bayesian Classifier) 就没有介绍。感兴趣的读者可以参考相关读物了解有关内容。

算法的介绍只是本书的一部分。在使用机器学习算法处理实际问题之前, 还需要进行如下步骤:

- (1) 数据探索 (data exploration);
- (2) 数据预处理 (data preprocessing);
- (3) 从原始数据中构建相应的特征, 即特征工程 (feature engineering)。

在实际使用机器学习处理数据的过程中, 数据探索和数据预处理是非常重要的步骤。通过数据探索, 我们可以了解数据的特性, 选用合理的预处理方法, 如处理缺失数据等。此外, 在很多情况下直接对原始数据使用机器学习算法并不能取得良好的效果, 我们需要对数据进行一些变换以生成新的特征, 这个过程称为特征工程。在实际使用机器学习算法解决问题时, 特征工程是非常重要的一步。良好的特征是成功应用机器学习的关键点之一。

在得到算法构建的模型后, 还需要评价和选择模型, 包括:

- 不同模型的评价标准;
- 从多个模型中选择最优模型的方法。

注意, 不同的算法有不同的评价标准, 如分类算法和回归算法的评价标准就不同。因此, 我们在介绍一类算法时, 通常会介绍此类算法的评价标准。例如, 介绍分类算法时, 我们介绍了准确率、AUC 等评价分类算法的标准以及在 R 中的计算方法, 同时也介绍了交叉检验和

R 中的 `caret` 包以帮助用户在 R 中进行模型选择。

此外，我们还介绍了 R 语言和必要的数学基础。本书广泛使用 R 语言介绍如何使用算法。R 语言具有免费、开放、简单易学的特点，在工业界的使用越来越广泛；同时，R 语言有大量免费的软件包可以使用，基本涵盖了机器学习的各个领域，本书所介绍的各个领域都能找到相应的 R 软件包。本书还介绍了常用的数学基础知识，包括概率统计、矩阵计算等，这样读者能够更加容易地理解和掌握算法的原理。

1.4.1 本书结构

本书大致可以分为两部分。前半部分介绍一些相关的基础知识，后半部分着重介绍各类算法。

由于我们在全书中都使用 R 来介绍如何使用各种机器学习算法，因此首先在第 2 章介绍 R 语言的基础知识。在第 3 章中，我们介绍相关的数学基础知识，包括概率统计的基础知识和矩阵计算的基础知识等。

第 4 章介绍数据的探索和预处理，包括数据类型、数据探索、数据预处理及数据可视化。

从第 5 章开始，我们着重介绍各类算法。我们首先从最基本的回归算法和分类算法开始讨论，然后介绍推荐算法和排序算法，最后介绍如何使用集成学习来综合多个模型以进一步提高模型的性能。

第 5 章介绍回归分析，包括常用的回归算法以及回归算法的评价和选取。我们从最基本的线性回归和最小二乘法开始讨论，然后讨论更加复杂的回归算法，包括岭回归、Lasso 和 Elastic Net。在介绍完多种回归算法之后，我们讨论如何评价和选取不同的回归算法。另外，我们讨论偏差-方差权衡 (bias-variance tradeoff)，并讨论模型复杂度 (model complexity) 的概念。最后，我们使用实际的案例分析来说明如何在 R 中使用和选取回归算法。

第 6 章讨论基本的分类算法，包括决策树、支持向量机和逻辑回归。我们引入了不同的损失函数 (loss function)，并讨论不同的分类算法如何对应不同的损失函数，以及如何使用正则化项 (regularization) 来控制模型的复杂度。与回归算法的讨论类似，我们也会讨论如何评价和选取不同的分类算法。为了解决实际中更复杂的分类问题，我们将会详细讨论如何解决不平衡分类 (imbalanced classification) 问题。本章还会介绍 R 中对应的软件包，这样读者可以直接尝试使用各种分类算法。另外，我们会着重介绍 `caret` 包，这样读者能够简单地使用交叉检验 (cross validation) 来为各种算法选取最优参数。

第 7 章介绍常用的推荐算法，主要包括基于相似度的推荐算法和基于矩阵分解的算法。我们还会介绍推荐算法的评价和选取，以帮助用户合理地选取算法。其中，基于相似度的算法包括基于内容的算法和基于邻域的算法；基于矩阵分解的算法包括：

- 无矩阵分解的基准方法；
- 基于奇异值分解 (SVD) 的推荐算法；
- 基于 SVD 推荐算法的变体，有 AFM 模型、翻转的 AFM 模型、ASVD 模型 (或者 SVD++ 模型)、翻转的 ASVD 模型、引入时间信息的模型。

基于内容的推荐算法不是第7章的重点。对于基于矩阵分解的诸算法，我们推导了对应的随机梯度下降（stochastic gradient descent）算法。对于基于邻域的推荐算法，我们将讨论基于用户的邻域推荐算法和基于商品的邻域推荐算法，并详细讨论基于邻域的推荐算法的核心部分：如何计算相似度和邻域。

第8章介绍排序算法，包括逐点排序（pointwise ranking）算法、逐对排序（pairwise ranking）算法和逐列排序（listwise ranking）算法。我们将着重介绍较实用的 LambdaMART 算法。

集成学习可以显著地提升多种算法的性能。在本书的最后一章，我们将介绍集成学习，包括基本思想和3类不同的集成学习方法：

- bagging 的基本思想及典型例子随机森林；
- boosting 的基本思想及典型例子提升决策树；
- stacking 的基本思想及应用。

同时，我们还会介绍 R 实现中流行的软件包，包括 randomForest 和 gbm。

1.4.2 阅读材料及其他资源

这里我们介绍机器学习及相关领域的一些教材和相关资源。首先介绍一下目前流行的多种关于机器学习的教材。

参考文献[1]是一本早期的关于机器学习的教材。该书是一本入门读物，介绍了机器学习的很多基本概念和算法。参考文献[2]是机器学习领域影响很大的一本教材，讨论了很多比较实用的算法，但该书对于读者的数学基础要求较高。参考文献[3]是参考文献[2]的简化版，内容和讲解上都更基础一些。参考文献[4]和参考文献[5]是两本从贝叶斯统计角度讨论机器学习的流行教材，阅读的难度稍大。参考文献[5]介绍的内容稍微前沿一些，很多都是从近年的论文中直接总结的。

参考文献[6]从使用 R 进行实际建模的角度介绍了机器学习，覆盖了 R 中的很多软件包和具体用法，讨论了实际建模中的各个步骤。但该书对很多算法没有具体讲解原理和步骤，仅适用于各种模型的入门。参考文献[7]也是近期出版的一本使用 R 来介绍机器学习的著作，该书以应用为主，所介绍的算法过于基础。

关于模式识别方面的教材，我们推荐参考文献[8]。该书是关于模式识别的经典教材，难度适中。

关于数据挖掘方面的教材，我们推荐参考文献[9]。该书讲解了基本的分类算法、关联规则和聚类算法。其他具有代表性的教材包括参考文献[10]和参考文献[11]。参考文献[10]是一本较早从数据库角度讨论数据挖掘的教材，其中介绍的算法比较基础。参考文献[11]则是一本以 WEKA^①为核心介绍数据挖掘的教材。此外，参考文献[12]的优点是贴近应用实际，使用实际中的一些应用例子来介绍算法。

^① <http://www.cs.waikato.ac.nz/ml/weka>

在掌握机器学习算法时，要注意深度和广度的问题。作为一名机器学习的实践者，熟知多种机器学习算法是必需的；同时，对于那些最常用的算法，要知道其基本思想和底层实现。本书将讨论机器学习的一些常用算法，但限于篇幅，很难做到面面俱到。读者可以根据自己的实际需要选择不同教材的相关章节阅读。

最前沿的关于机器学习的研究文章通常会在机器学习领域的顶级会议和期刊上发布或发表。此类顶级会议主要包括：

- Annual Conference on Neural Information Processing Systems (NIPS)
- International Conference on Machine Learning (ICML)
- ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)

机器学习领域的顶级期刊主要包括：

- 《IEEE Transactions on Pattern Analysis and Machine Intelligence》(TPAMI)
- 《Journal of Machine Learning Research》(JMLR)
- 《Machine Learning Journal》

这里我们只列出了部分顶级的会议和期刊。感兴趣的读者可以在互联网上找到更多的此类会议和期刊。注意，中国计算机学会在其官网上给出了各领域的推荐会议和期刊目录^①，我们给出的列表主要涉及其中的人工智能、数据挖掘两个领域。

对于机器学习算法的广大应用者来说，最重要的问题是如何利用机器学习算法来解决实际问题。例如，如何将实际问题转化为一个能够直接应用机器学习算法的问题。网站 kaggle.com 有很多关于机器学习的竞赛，是一个提高这方面能力的优秀媒介。例如，前面讨论的 Heritage Health Prize 就是由 kaggle.com 组织的。kaggle.com 还提供了过去竞赛的很多获胜方案。在解决具体实际问题时，如果问题和 kaggle.com 中的某个竞赛类似，则可以借鉴获胜方案。互联网上也有关于 kaggle.com 竞赛的介绍，例如，SlideShare 网站^②就给出了一个很好的关于 kaggle.com 的介绍。

关于机器学习的实际应用工具，除了 R 之外，比较常用的还有 Python 中的 [scikit-learn](http://scikit-learn.org)^③。该软件包涵盖了机器学习中的很多实用算法，包括分类、回归、聚类、数据降维和预处理等。该软件包的文档极为完备。如果读者经常使用 Python，[scikit-learn](http://scikit-learn.org) 是一个极好的机器学习库。WEKA 也是数据挖掘领域使用较多的一个软件包，该软件包集成了很多常用的机器学习算法，同时也提供了调用的 API。当数据规模较大时，很多时候我们需要使用 Hadoop 平台上的 Mahout 库^④和 Spark 平台上的 MLlib 库^⑤。在本书中，为了方便读者简单地使用各种算法，我们以 R 为基础来介绍各种算法的使用。

① <http://www.ccf.org.cn/sites/ccf/paiming.jsp>

② <http://www.slideshare.net/ksankar/oscon-kaggle20>

③ <http://scikit-learn.org/stable/>

④ <http://mahout.apache.org/>

⑤ <https://spark.apache.org/docs/1.1.0/mllib-guide.html>

第2章 R语言

2.1 R的简单介绍

R 是一种自由、免费、开源的解释型编程语言，支持 Unix、Windows、Mac 等系统平台。与 Perl、Python、JavaScript 等其他解释型语言相比，R 具有强大的数据分析工具和图形工具，支持多种设备上的分析和展示。

R 语言的历史可以追溯到 1976 年贝尔实验室开发的一种用于数据探索、统计分析、作图的解释型编程语言——S 语言。S 系统由 Richard A. Becker、John M. Chambers 等人实现，它无须用户关心内存分配与数据结构细节，具有良好的可移植性与可扩展性^[13]。20 世纪 80 年代末期，Richard A. Becker、John M. Chambers、Allan R. Wilks 等人对 S 语言进行了功能方面的更新，参见参考文献[14]。1998 年，为了表彰 S 系统在分析、可视化和操作数据方面的贡献，ACM 授予其主要设计者 John M. Chambers 软件系统奖^①。在 S 语言的基础上，MathSoft 公司^②研发了商业软件 S-Plus，其特点有二：一是可以交互式地挖掘数据中的信息，并轻松实现新的统计方法；二是可以直接使用 Excel、Lotus、Access、SAS、SPSS 等软件的数据，具有极好的兼容性。有关 S-Plus 的数据结构、函数、作图以及面向对象编程，参见参考文献[15]。

1975 年，MIT 人工智能实验室为 Lisp 类语言研发了解释器 Scheme^[16]。奥克兰大学的 Ross Ihaka、Robert Gentleman 等人综合了 Scheme 与 S 语言的优点研发出了 R 语言^[17]。R 语言与 S 语言非常类似，但其底层实现和语法都来自 Scheme。具体实现方面，Ross Ihaka、Robert Gentleman 等人首先为 Scheme 的一个子集编写了解释器，然后分如下 3 个阶段向 S 语言靠拢：一是替换语言的语法解析器，使得表面上看来语法与 S 类似，但底层语法仍来自 Scheme；二是用 S 的向量数据类型替换原有的标量数据类型，这一步改动较大；三是引入 S 语言中对参数延迟求值的思想。最终获得的 R 语言在功能上弱于 S-PLUS，但同样具备很强的实用性——根据参考文献[17]，R 在可移植性、计算效率、内存管理等方面都具有优势。

R 语言的得名主要有两方面的原因，一方面是为了感谢 S 语言的正面影响，另一方面是因为两位主要研发者名字的首字母都为 R^[17]。有关 R 的文档资源可以从 <https://cran.rstudio.com/manuals.html> 获取，各种操作系统环境下的常见问题可以到如下页面寻求解答：<https://cran.rstudio.com/faqs.html>。这里 CRAN 是 Comprehensive R Archive Network 的简称，意为 R 语言的综合网络资源所在。

① http://awards.acm.org/award_winners/chambers_6640862.cfm

② 2001 年更名为 Insightful 公司，2008 年被 TIBCO 收购，后者于 2014 年被 Vista Equity Partners 收购。

由于其免费、开放及实用的特性，R 在统计学、机器学习和数据科学（data science）中越来越流行。很多算法都以第三方软件包的形式提供。在 CRAN 上，2016 年 4 月时已有超过 8000 个软件包^①，涵盖了统计学、机器学习、生物统计学等众多领域的大量算法。本书中介绍的大量机器学习算法在 R 中都有成熟的实现，因此本书的实例均基于 R 语言给出。在本章中，主要介绍 R 的下载、安装、基本语法以及软件包的使用。

2.2 R 的初步体验

首先访问 <https://www.r-project.org/>，下载安装 3.0 或更高版本的 R。以 Windows 下的 3.2.3 版本为例，R 图形界面启动后会给出如图 2-1 所示的命令行控制台，首先介绍 R 的版本、版权、贡献者及简单使用方法，最后给出默认的命令提示符“>”。

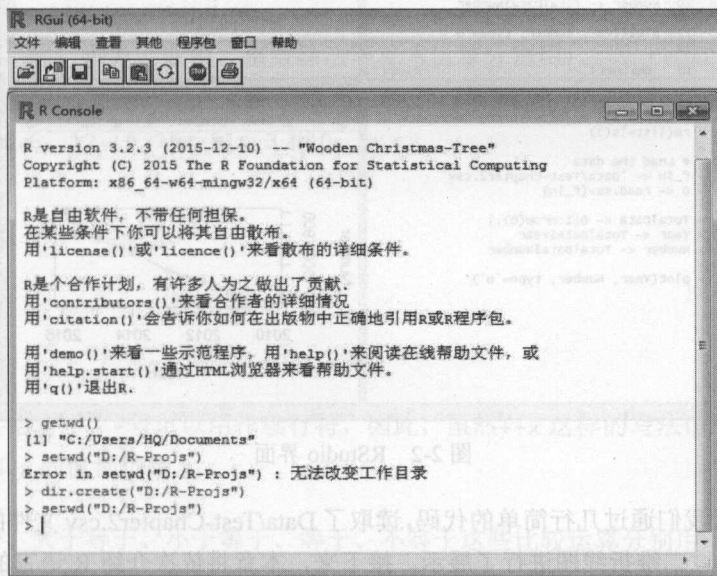


图 2-1 R 图形界面及命令行控制台

接下来，可以像使用 MS-DOS 系统一样，在控制台中编写 R 语言程序。例如，图 2-1 给出了获取工作目录的命令 `getwd()`、创建文件夹的命令 `dir.create()` 及修改工作目录的命令 `setwd()`。但这样逐条输入命令毕竟比较麻烦，可以通过“文件”菜单下的“新建程序脚本”来编写 R 程序，通过“打开程序脚本”来读入已保存的 R 程序。

为了方便地编写和调试 R 程序，可以从 <https://www.rstudio.com/products/RStudio/> 下载免费、开源的 R 语言集成开发环境——RStudio。RStudio 有两个版本可供选择：单机版 RStudio

^① <https://cran.r-project.org/web/packages/>

Desktop 和服务版 RStudio Server, 我们选择 RStudio Desktop。图 2-2 给出了 Windows 下 0.99.491 版本的 RStudio 运行界面, 包括菜单、内容区、命令行控制台、工作区环境、展示区等。其中, 内容区不仅可以显示源代码, 还可以给出指定数据结构的内容; 命令行控制台可以提示程序中的错误或输出中间结果; 工作区环境简要地对当前运行环境中的数据、变量、函数等进行描述; 展示区列出文件目录、输出图表等信息。

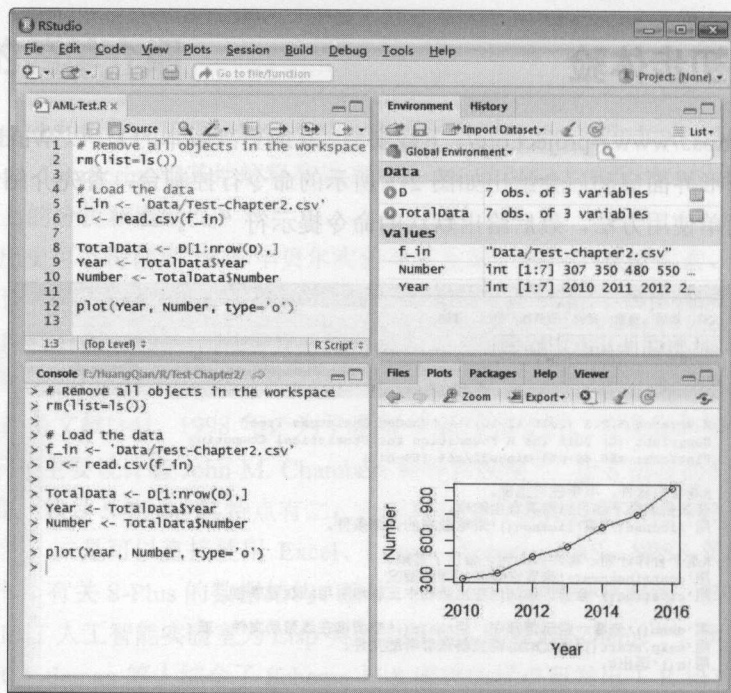


图 2-2 RStudio 界面

在图 2-2 中, 我们通过几行简单的代码, 读取了 Data/Test-Chapter2.csv 文件的 Year、Number 两列数据, 并通过二维折线图进行了展示。接下来, 本章将依次介绍 R 语言的基本语法、常用数据结构、使用技巧和软件包。

2.3 基本语法

从图 2-2 的例子可以看出, R 语言允许直接使用变量, 不需要预先定义数据类型。本节主要介绍 R 的语句和函数。

2.3.1 语句

本节主要介绍 R 语言的注释语句、表达式语句、函数调用语句和控制语句。

1. 注释语句

以符号#开头的语句称为注释语句。注释内容占用多行时，每行前面都需要加#。在 RStudio 中，可以使用快捷键 Ctrl+Shift+C 来注释一块程序。对于已经注释的块程序，也可以使用快捷键 Ctrl+Shift+C 来去除注释。

2. 表达式语句

在 R 中，所有的变量、数据及函数都以对象（object）的形式保存在内存中。对象的名字必须以字母开头（大小写皆可），中间可以包含数字、点（.）及下划线（_）。例如，a3.b、a3_b 都是合理的对象名，而_a3 则不是。此外，R 是对大小写敏感的，因此 a 和 A 是不同的对象。

赋值功能可以用=、<-或->实现。例如，下面 3 条语句分别将变量 x、y 和 z 赋值为 3、4 和 3.4：

```
x<-3
4->y
z=3.4
```

加、减、乘、除、乘方、取模、整数除法等算术运算分别用+、-、*、/、^、%%和%/实现。例如，在以上赋值的基础上，下面的语句执行之后，x0、x1、x2、x3、x4、x5、x6 的值分别为 6.4、0.6、12、0.75、81、3 和 0。

```
x0<-x+z
x1<-y-z
x2<-x*y
x3<-x/y
x4<-x^y
x5=x%%y
x6=x%/y
```

注意，在 R 语言中，+号可以用作续行符，因此，虽然++x 这样的写法语法正确，但如下赋值之后，x7 和 x 的值都仍是 3：

```
x7=++x
```

大于、小于、大于等于、小于等于、等于、不等于这些比较运算分别用>、<、>=、<=、==和!=表示。比较运算的结果为 TRUE（1）或 FALSE（0）。例如，字符型数据可以用单引号或者双引号分别表示为：

```
s1 = 'hello world'
s2 = "hello world"
```

这里 s1 和 s2 是一样的，因此 s1==s2 返回 TRUE。

与、或、非、异或这些逻辑运算分别用&、|、!和 xor 实现。例如，xor(x,y) 在 x 和 y 的逻辑值不同时计算得到 TRUE，在 x 和 y 的逻辑值相同时计算得到 FALSE。对于后面将要介绍的向量数据类型，x&y 只对向量 x 和向量 y 的第一个元素进行逻辑“与”运算，x|y 只对向量 x 和向量 y 的第一个元素进行逻辑“或”运算。假定向量 x 和 y 长度一致，则 x&y 对向量 x 和向量 y 的每个元素进行逻辑“与”运算，结果是一个相同长度的向量；类似地，x|y 对

向量 x 和向量 y 的每个元素进行逻辑“或”运算。此外, $!x$ 和 $xor(x,y)$ 也都返回一个向量。

此外, 使用 $a \%in\% b$, 可判断 a 是否在 b 中间。例如, $1 \%in\% 1:5$ 是 TRUE, $1 \%in\% 2:5$ 是 FALSE。

3. 函数调用语句

函数调用语句的一般形式为:

函数名(实际参数 1, 实际参数 2, ...)

与其他程序设计语言类似, R 语言中调用函数的过程也是一个把实际参数赋给函数定义中的形式参数, 然后执行函数体并返回函数值的过程。图 2-1 和图 2-2 中给出的 `getwd()`、`dir.create()`、`setwd()`、`rm()` 和 `plot()` 都是函数调用语句; `D<-read.csv()` 是函数调用语句和赋值语句组成的复合语句。

4. 控制语句

控制语句用于 R 程序的执行流程, 主要包括条件判断语句、循环语句、跳转语句等。本节讨论的条件判断语句包括 `if`、`if...else` 等, 循环语句包括 `for`、`while`、`repeat` 等, 跳转语句包括 `break` 等。在 R 中, 最常用的循环语句是 `for` 和 `while`, 下面给出示例代码。

示例 1:

```
n = 10
for(i in 1:n){
  if (i%%2)
    k = 1
  else
    k = 0
  cat("k=", k)
  cat("\n")
}
```

对应的输出如下:

```
k= 1
k= 0
k= 1
k= 0
k= 1
k= 0
k= 1
k= 0
k= 1
k= 0
```

示例 2:

```
n = 10
s = 0
while(n>0){
```

```
s <- s + n
n <- n-1
if (n==3)
  break
}
```

当循环结束时, $s=49$ 。

2.3.2 函数

本节介绍一些常用的函数。在后面的章节中, 我们会根据需要介绍更多的函数。

1. 基本函数

我们把安装 R 之后即可使用的函数称为基本函数, 如图 2-1 中的 `getwd()`、`dir.create()`、`setwd()` 等。本节介绍部分基本函数的常见用法。

(1) `ls()` 函数。函数 `ls()` 的功能是列出内存中的所有对象名, 包括数据、函数等。例如, 对于 2.3.1 节的示例 1, 函数 `ls()` 的调用结果为:

```
[1] "i" "k" "n"
```

当对象较多时, 可以增加正则表达式模式参数 `pat` 以实现更好的匹配。例如, 可以用 `ls(pat="u")` 来指定名称中包含字符 `u` 的对象, 或者用 `ls(pat="^s")` 来指定名称以字母 `s` 开头的对象。若想获取对象的类型、值等信息, 可调用函数 `ls.str()`, 此时上述 `pat` 参数同样适用。

(2) `help()` 函数。读者可以调用帮助函数 `help()` 获取关于函数的进一步介绍信息。例如, 若想获得函数 `plot` 的帮助信息, 可调用 `?plot`、`help(plot)` 或 `help("plot")`。

(3) `rm()` / `remove()` 函数。我们在图 2-2 中调用的第一个函数就是 `rm(list=ls())`, 其功能是删除内存中的所有对象。我们可以借助上述 `pat` 参数实现选择性删除, 也可以通过 `rm(k)`、`rm(i,n)`、`rm(i,k,n,sum)` 等调用对内存中的一个或多个对象进行点名删除。`remove()` 函数的使用与 `rm()` 函数完全相同。

(4) `paste()` 函数和 `paste0()` 函数。我们通过一个例子展示一下 `paste()` 函数的功能:

```
paste('hello', 'world', sep='-')
```

对应的输出是 "hello-world"。`paste()` 函数的作用就是将多个字符型对象串起来, 其中 `sep` 参数指定串起来时的分隔符。在 `paste` 函数中, `sep` 的默认值为 " "(空格字符)。使用 `paste()` 函数时, 输入的字符串对象可以是多个:

```
paste('hello', 'world', 'from', 'R', sep='-')
```

返回的结果是 "hello-world-from-R"。下面给出一个更复杂的关于 `paste()` 函数的例子:

```
paste(c("x", "y", "z"), 1:7, sep="-")
```

这里 `paste()` 函数的第一个参数只有 3 个元素, 因此会循环补足以便与第二个参数中的 1~7 相

匹配。注意, 1:7 并不是字符型对象, `paste()` 函数会将其转换为字符型对象。所得到的结果为:

```
[1] "x-1" "y-2" "z-3" "x-4" "y-5" "z-6" "x-7"
```

如果我们将 `sep` 设为 "" (空字符), `paste(c("x","y","z"),1:7,sep="")` 对应的输出为:

```
[1] "x1" "y2" "z3" "x4" "y5" "z6" "x7"
```

很多时候, 我们希望分隔字符为空, 这时推荐使用 `paste0` 函数。例如, `paste0(c("x","y","z"),1:7)` 的输出为:

```
[1] "x1" "y2" "z3" "x4" "y5" "z6" "x7"
```

可以看出, 上述调用与在 `paste` 函数中将 `sep` 设为空字符的效果是一样的。当然, 在 `paste0` 函数中, 我们也可以显式地指定 `sep`。

(5) `cat()` 函数。以用户自定义的形式输出结果。例如, 前面我们在输出中间结果时, 用

```
cat("k=",k)
cat("\n")
```

实现了拼接输出和换行, 获得了比 `print()` 函数更好的输出效果。

(6) `plot()` 函数。使用函数 `plot(x,y)` 可以作图画出 y 相对于 x 变化的情况。参数 `type` 可以用来指定图中点的形状。假设我们使用向量 `Year` 记录年份, 使用向量 `Number` 记录每年对应的数据, 则使用

```
plot(Year, Number, type='o')
```

可以获得与图 2-2 右下角类似的折线图。注意, 如果生成绘图后继续调用 `plot` 函数, 通常会覆盖之前的绘图。

(7) `c()` 函数。该函数的功能是把若干个参数组合成一个向量或者列表, 函数的返回值即为组合的结果。例如, 下面的语句把 4 个值组合为向量 `x`:

```
x <- c(1, 0, 2, 9)
```

利用 `c()` 函数, 也能将多个 `x` 连接起来。例如:

```
x1 <- c(1,0,2,9)
x2 <- c(4,3)
x3 <- c(x1, x2)
```

则 `x3` 为:

```
[1] 1 0 2 9 4 3
```

2. 自定义函数

R 语言允许以如下的形式自定义函数:

```
函数名 <- function(形式参数1,形式参数2,...){
  函数体
}
```

函数调用的方式见 2.3.1 节。

例如，下面的代码首先定义了一个求和函数，然后以 3、4 作为实际参数调用该函数：

```
aml_sum <- function(a, b) {
  r <- a+b
  return (r)
}
x<-3
y<-4
z<-aml_sum(x,y)
print(z)
```

程序的输出为：

```
[1] 7
```

注意，上面对 r 的赋值仅在函数定义的内部可见，在程序的其他位置 r 是没有定义的。采用如下的超级赋值（super assignment）运算符可以使 r 从局部变量变为全局变量：

```
r <<- a+b
```

在前面的函数定义中， $\{$ 和 $\}$ 分别表示函数体的开始与结束。 $\{$ 写在第一行的行末仅是一种书写风格，亦可单独成行书写为如下形式：

```
函数名 <- function(形式参数 1, 形式参数 2, ...)
{
  函数体
}
```

如果函数体只有一条语句，大括号可以省略。

2.4 常用数据结构

数据结构指的是程序设计语言所支持的数据组织方式。前面提到，R 语言中把操作的实体称为“对象”，因此本书所讨论的数据结构，实际上就是“对象”的数据组织方式，主要包括向量（vector）、因子（factor）、矩阵（matrix）、数组（array）、数据框（data frame）和列表（list）等。其中向量与列表的概念相似，但向量要求其组成元素的类型相同，而列表对此不作要求，可视为“泛化的向量”。数组是一个 k 维的数据表格，而矩阵是数组的二维特例。在实际数据处理中，我们一般使用矩阵表示数据即可。本节着重介绍向量、因子、矩阵、数据框和列表。

2.4.1 向量

1. 向量和基本数据类型

首先，R 中的基本类型（base type）是向量（vector），而不是标量（scalar）。在 R 中，向量由同一类型的若干元素组成，而且我们可以使用下标系统来访问每个元素。注意，在 R 中，下标都是从 1 开始的。根据存储元素类型的不同，向量可以分为如下几类：

- 整型 (integer);
- 数值型 (numeric);
- 字符型 (character);
- 逻辑型 (logical);
- 复数型 (complex)。

严格地讲, 整型是数值型的一个特例。在实际数据处理中, 复数型很少遇到, 因此我们主要讨论前面的 4 种类型。

下面 4 条语句分别得到逻辑型向量 w 、整型向量 x 、数值型向量 y 和字符型向量 z :

```
w <- c(TRUE, FALSE, TRUE)
x = 1:9
assign("y", c(2.5, 4.2, 1.8))
z <- c("m1", "a2", "c3", "h4", "i5", "n6", "e7")
```

这里 `assign` 函数可替换为 `<-`。注意, 在同一向量中不能包含不同类型的元素。这里可以使用 `class()` 函数来检查生成的这 4 个向量的类型:

```
> class(w)
[1] "logical"
> class(x)
[1] "integer"
> class(y)
[1] "numeric"
> class(z)
[1] "character"
```

在 R 中, 逻辑值的表示为 `TRUE` 和 `FALSE`, 也可以简写为 `T` 和 `F`。下面两条语句的效果是完全一样的:

```
w1 <- TRUE
w1 <- T
```

可以直接使用下标来访问向量中的一个或者多个元素。如果要访问向量 y 中的第 2 个元素, 可直接使用 `y[2]`。例如, 语句

```
print(y[2])
```

将输出

```
[1] 4.2
```

使用 `y[1:2]` 可以访问 y 中的前两个元素。`print(y[1:2])` 的输出如下:

```
[1] 2.5 4.2
```

此外, 使用前面介绍的 `c()` 函数可以将若干个向量组成一个新的向量。例如:

```
p <- c(1/y, y)
print(p)
```

将输出

```
[1] 0.4000000 0.2380952 0.5555556 2.5000000 4.2000000 1.8000000
```

前面在介绍控制语句时，我们用过如下语句：

```
for(i in 1:n){...}
```

这里 1:n 就是从 1 到 n 步长为 1 的整数向量。如果改为 n:1 的形式，则循环体中 i 从 n 开始依次递减到 1，即从 n 到 1 步长为 -1 的整数向量。具体来说，v1:v2 生成如下序列：

- 如果 $v1 > v2$ ，则步长为 -1，生成的向量为 $v1, v1-1, v1-2, \dots, v2$ ；
- 如果 $v1 < v2$ ，则步长为 1，生成的向量为 $v1, v1+1, v1+2, \dots, v2$ 。

这里我们简单讨论一下生成向量的最后一个元素。当步长为正时，最后一个元素是小于等于 $v2$ 的最大值；当步长为负时，最后一个元素是大于等于 $v2$ 的最小值。例如，1.5:4 生成的向量为 c(1.5, 2.5, 3.5)，4.5:1 生成的向量为 c(4.5, 3.5, 2.5, 1.5)。

更一般的序列可以使用 seq() 函数生成。例如，我们可以使用 seq(1, 9, by=2) 生成 1~9 的奇数，这里第一个参数是起点，第二个参数是终点，参数 by 指定了步长。

在实际中，很多时候需要先生成一定长度的向量，再逐步修改其中的元素。在这种情况下，我们可以使用 rep 函数生成一个所有元素都相同的向量。下面的代码产生了 3 个长度为 5 的向量(但类型不同)，其中 v_ch 中每个元素都是空字符，v_num 中每个元素都是 0，v_logic 中每个元素都是 FALSE。

```
v_ch <- rep('', 5)
v_num <- rep(0, 5)
v_logic <- rep(FALSE, 5)
```

其对应的输出为：

```
> v_ch
[1] "" "" "" "" ""
> v_num
[1] 0 0 0 0 0
> v_logic
[1] FALSE FALSE FALSE FALSE FALSE
```

上面的 v_ch、v_num 和 v_logic 可以用下面 3 个函数得到，效果完全一样：

```
v_ch <- character(5)
v_num <- numeric(5)
v_logic <- logical(5)
```

这 3 个函数的输入参数表示生成向量的长度。

在实际中，很多时候还需要显式地进行类型转换。常用的类型转换函数有：

- as.integer
- as.numeric
- as.character

- `as.logical`

如 `as.numeric('3.4')` 返回 3.4, `as.logical(1)` 返回 TRUE, `as.logical(0)` 返回 FALSE, `as.character(3.4)` 返回 "3.4", `as.integer('3.4')` 返回 3。

在 R 中, 无论数据的类型如何, 缺失值都使用 NA 表示。对 NA 进行任何操作所得的结果都是 NA。例如, `NA+1`、`NA>0` 等的结果都是 NA。对于 NA 值, 我们一般使用函数 `is.na()` 来处理。下面是一个简单的例子。

```
> w <- NA
> is.na(w)
[1] TRUE
```

由于 w 是 NA, 因此 `is.na(w)` 输出值为 TRUE。

2. 向量的运算

(1) 基本运算。向量的基本运算直接施加在其组成元素上。例如, 语句

```
x <- c(16, 33, 45, 88)
y <- x/2
print(y)
```

将得到如下输出:

```
[1] 8.0 16.5 22.5 44.0
```

在上述算术赋值的基础上, 赋值语句

```
large <- x>17
```

可以得到值为 (FALSE, TRUE, TRUE, TRUE) 的逻辑向量 large。

(2) 常用函数。对于给定的向量 x, 我们列出 R 中提供的常用函数:

- `length(x)` 返回其元素个数;
- `sum(x)` 计算其元素的和;
- `prod(x)` 计算其元素的乘积;
- `max(x)` 和 `min(x)` 分别计算其中元素的最大值与最小值;
- `mean(x)` 计算 x 中元素的平均值;
- `median(x)` 计算 x 中元素的中位数;
- `which.min(x)` 返回 x 中最小元素所对应的下标;
- `which.max(x)` 返回 x 中最大元素所对应的下标;
- `which(x>a)` 返回 x 中大于 a 的元素所对应的下标。在 `which` 函数中, 用户可以设定不同的条件得到不同结果;
- `unique(x)` 返回 x 中所有不同的取值;
- `range(x)` 返回 x 中的最小值和最大值, 等同于 `c(min(x), max(x))`;
- `rev(x)` 将 x 中所有元素的顺序反转, 如 `rev(1:9)` 的结果是 9:1;
- `sort(x)` 将 x 中元素按照升序排列。`sort(x, decreasing=TRUE)` 可将 x 中元素按照降序排列。

除此之外, 还可以使用 `log`、`exp`、`sin`、`cos`、`tan`、`sqrt`、`var` 等数学函数。例如, 如下语句首先对向量中的元素由大到小排序, 然后计算每个元素的平方根:

```
x <- c(36, 16, 49, 81)
sqrt(sort(x, TRUE))
```

对应的输出如下:

```
[1] 9 7 6 4
```

2.4.2 因子

在 R 中, 因子是用来表示分类变量 (categorical variable) 的一种有效方法。所谓分类变量就是取值来自一个集合的变量。顾名思义, 分类变量的每一个取值表示该变量的一个状态 (或者分类)。一个典型的例子就是分类问题中的类别。用 R 处理分类问题时, 样本的类别信息通常以因子的形式保存。下面我们用一个分类变量的例子来解释因子数据。假设有一个向量 `gender_v` 用来描述一组对象的性别:

```
gender_v <- c('Male', 'Female', 'Female', 'Male')
```

如果将每个元素都存储为字符型数据, 则会占用比较大的存储空间。为节省存储空间, 可以用正整数的形式保存每个元素, 并保存正整数到各个取值之间的映射关系。这样得到的数据就是因子数据, 而所有不同的取值则称为水平 (level)。在实际中, 水平值一般是比较长的字符, 而我们只保存整数则可以显著地节省保存空间。

下面我们使用具体的例子来说明因子数据。首先将 `gender_v` 转化为因子数据:

```
f_gender <- factor(gender_v)
```

在 R 中打印 `f_gender` 和 `gender_v`, 可以直接看出它们的不同:

```
> gender_v
[1] "Male" "Female" "Female" "Male"
> f_gender
[1] Male Female Female Male
Levels: Female Male
```

注意, `f_gender` 保存的并不是字符型数据 (输出中没有引号)。此外, `Levels` 还列出了所有不同的水平值。

通过使用函数 `levels`, 可以得到 `f_gender` 对应的所有水平值, 下面是对应的输出:

```
> levels(f_gender)
[1] "Female" "Male"
```

还可以直接将 `f_gender` 转化为数值型数据, 对应的输出为:

```
> f_gender_num <- as.numeric(f_gender)
> f_gender_num
[1] 2 1 1 2
```

可以看出 `f_gender_num[1]` 为 2，表示其值对应于 `levels(f_gender)[2]`，也就是 'Male'。我们可以调用 `as.character()` 函数直接得到 `f_gender` 中每个元素对应的字符：

```
> f_gender_ch <- as.character(f_gender)
> f_gender_ch
[1] "Male" "Female" "Female" "Male"
```

2.4.3 矩阵

1. 矩阵的定义

在介绍矩阵之前，首先引入“维度”的概念。对于 R 语言中的对象，可以通过在 `[]` 中指定下标的方式访问其中的某个元素。例如，在下面的例子中，可以通过 `x[2]` 访问向量 `x` 的第 2 个元素。

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8)
```

如果 `[]` 中只能指定一个下标，则称该对象是一维的；如果 `[]` 中最多只能指定两个下标，则称该对象是二维的，依此类推。在上面的例子中，我们不能直接将 `x` 视为二维向量，通过 `x[2,4]` 或 `x[4,2]` 访问其第 8 个元素。

R 语言允许用函数 `dim` 来设定对象的维度。例如，语句

```
dim(x) <- c(2,4)
print(x)
```

将产生如下输出：

```
      [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

从而我们可以将 `x` 看作二维向量，并分别用 `x[2,3]`、`x[1,4]` 访问元素 6 和 7。本书把数值或复数型的二维向量称为矩阵^①。

除上述方法外，还可以利用 `array` 函数或 `matrix` 函数来得到矩阵。例如，上述矩阵可通过如下两种函数调用之一来构造：

```
array(1:8, dim=c(2,4))
matrix(1:8, nrow=2, byrow=FALSE)
```

这里可以使用 `array` 函数来创建更高维的数组。例如，下面的代码产生了一个三维数组：

```
array(1:27, dim=c(3,3,3))
```

因为本书主要讨论矩阵，所以我们主要使用 `matrix` 函数。例如，创建一个 4×3 的零矩阵可以使用：

```
M <- matrix(0, nrow=4, ncol=3)
```

^① 有些资料（如参考文献[18]和[19]）引入“数组”来定义矩阵，但 R 语言并未严格区分向量、数组和矩阵。

或者简写为:

```
M <- matrix(0, 4, 3)
```

在 R 中, 矩阵的每行或者每列都可以有相应的名字。使用 `colnames(M)` 可以获取和修改 M 的列名, 使用 `rownames(M)` 可以获取和修改 M 的行名:

```
> colnames(M)
NULL
> rownames(M)
NULL
```

这里由于没有预先设置, 因此列名和行名都为空。下列语句将 M 的列名分别改为 'F1'、'F2' 和 'F3':

```
> colnames(M) <- c('F1', 'F2', 'F3')
```

这样 M 就有了列名:

```
> M
      F1 F2 F3
[1,]  0  0  0
[2,]  0  0  0
[3,]  0  0  0
[4,]  0  0  0
```

在实际使用矩阵或稍后讨论的数据框来处理数据时, 基本上每行对应一个样本, 每列对应一个变量, 为每列赋予一个列名有助于更好地处理数据。

2. 矩阵的运算

(1) 基本运算。对于两个矩阵 A 和 B, 其矩阵乘积在 R 中用 `A %*% B` 表示。例如, 下面的代码将 2 行 4 列的矩阵 x 与 4 行 2 列的矩阵 y 相乘:

```
x <- array(c(1, 2, 3, 4, 5, 6, 7, 8), c(2,4))
y <- array(c(1, 2, 3, 4, 5, 6, 7, 8), c(4,2))
z <- x %*% y
print(z)
```

获得的输出为:

```
      [,1] [,2]
[1,]   50  114
[2,]   60  140
```

此外, 可以分别使用 `A+B`、`A-B` 来实现加法和减法。

在上面的例子中, 如果已知 x 和 z, 则求解 y 的过程是求矩阵积的逆过程, 可通过如下函数调用实现:

```
y <- solve(x, z)
```

注意, 直接调用 `solve(x)` 将得到 x 的逆矩阵 x^{-1} , 但不推荐用如下方式计算 y:

可以 `y <- solve(x) %*% z` 为 2, 表示其值对应于 `levels(f.gender)[2]`, 也就是男。

(2) 常用函数。假设用 A 和 B 表示两个矩阵, 则

- `t(A)` 表示 A 的转置矩阵;
- `crossprod(A, B)` 计算 `t(A)` 与 B 的矩阵积;
- `tcrossprod(A, B)` 计算 A 与 `t(B)` 的矩阵积;
- `diag(A)` 返回矩阵 A 的对角元素;
- `cbind(A, B, ...)` 函数将矩阵 (或者向量) A、B 沿水平方向结合起来, A 和 B 中的每一列都是结合之后矩阵的列, 函数名中的 'c' 表示 column;
- `rbind(A, B, ...)` 函数矩阵 (或者向量) A、B 沿竖直方向结合起来, A 和 B 中的每一行都是结合之后矩阵的行, 函数名的 'r' 表示 row;
- `nrow(A)` 返回矩阵 A 的行数;
- `ncol(A)` 返回矩阵 A 的列数;
- `dim(A)` 同时返回矩阵 A 的行数和列数;
- `rowMeans(A)` 计算矩阵 A 中每一行的均值, 返回结果是一个向量;
- `rowSums(A)` 计算矩阵 A 中每一行的和, 返回结果是一个向量;
- `colMeans(A)` 计算矩阵 A 中每一列的均值, 返回结果是一个向量;
- `colSums(A)` 计算矩阵 A 中每一列的和, 返回结果是一个向量;
- `det(A)` 计算方阵 A 的行列式;
- `eigen(A)` 计算方阵 A 的特征值与特征向量;
- `head(A, k)` 返回矩阵 A 的前 k 行;
- `tail(A, k)` 返回矩阵 A 的最后 k 行。

相关数学概念参见第 3 章。

2.4.4 数据框

1. 数据框的定义

数据框 (data frame) 是 R 语言中最接近 SAS 和 SPSS 数据集的数据结构, 它在形式上类似于矩阵, 但允许各列的数据类型不同, 可以很方便地表示实际中的各种数据, 因此是 R 中最常用的数据结构之一。

下面我们通过读取文件和手工创建两种方式来定义数据框。

一方面, 我们可以通过读取文件的方式创建数据框:

```
D <- read.csv('Data/Test-Chapter2.csv')
print(D)
```

获得的输出为:

```
Year Volume Number
1 2010    3000    307
```

```

2 2011 3500 350
3 2012 4000 480
4 2013 4500 550
5 2014 5000 700
6 2015 5500 800
7 2016 6000 1000

```

另一方面,我们也可以手工创建数据框:

```

Year <- 2010:2016
Volume <- c(3000,3500,4000,4500,5000,5500,6000)
Number <- c(307, 350, 480, 550, 700, 800, 1000)
df2 <- data.frame(Year, Volume, Number)
print(df2)

```

获得的输出为:

```

  Year Volume Number
1 2010  3000    307
2 2011  3500    350
3 2012  4000    480
4 2013  4500    550
5 2014  5000    700
6 2015  5500    800
7 2016  6000   1000

```

2. 数据框的操作

(1) 基本操作。数据框中的元素可以按列名或按坐标访问。例如,对于前面定义的数据框 df2,我们可以通过 df2[1:3,2:3] 访问其第 2~3 列的前 3 行,也可以通过 df2\$Volume、df2[[2]] 或 df2[, 'Volume'] 访问其第 2 列。2.4.5 节将进行更加详细的讨论。

(2) 常用函数。对于给定的数据框 dfa 和 dfb, rbind(dfa, dfb) 表示按 dfa 中的列顺序依次捆绑 dfa 和 dfb 的行,生成一个新的数据框。下面我们使用具体的例子来说明 rbind 的使用。首先我们产生两个数据框 df3 和 df4:

```

No. <- c(4001:4005)
Score <- c(82,77,90,63,85)
Age <- c(20, 19, 21, 22, 20)
df3 <- data.frame(No., Score, Age)
No. <- c(4006:4007)
Score <- c(92,65)
Age <- c(20, 19)
df4 <- data.frame(Score, Age, No.)

```

使用 rbind(df3, df4) 得到的输出为:

```

  No. Score Age
1 4001    82  20
2 4002    77  19

```



```

3 4003    90  21
4 4004    63  22
5 4005    85  20
6 4006    92  20
7 4007    65  19

```

注意, 上述代码中 df4 与 df3 的列顺序不同。类似地, cbind(dfa,dfb) 表示按 dfa 中的行顺序依次捆绑 dfa 和 dfb 的列, 生成一个新的数据框。例如, 在上述代码的基础上, 代码

```

Grade <- c('B','C','A','D','B')
df5 <- data.frame(Grade)
cbind(df3,df5)

```

得到的输出为:

```

  No. Score Age Grade
1 4001    82  20    B
2 4002    77  19    C
3 4003    90  21    A
4 4004    63  22    D
5 4005    85  20    B

```

可见, df3 的值并没有因为之前的 rbind(df3,df4) 调用而改变。此外, 调用 cbind 函数时, 也可以直接以 Grade 作为第二个参数, 结果一样。

利用 colnames 和 rownames 可以访问或者改变数据框的列名和行名。利用 head 和 tail 函数, 可以探索数据框的前面和最后若干行。这些函数的使用可参见矩阵部分的相应讨论。

利用 expand.grid 函数, 可以将参数的不同组合保存在一个数据框中。例如:

```

age_v <- 6:8
gender_v <- c('Male', 'Female')
D <- expand.grid(age=age_v, gender=gender_v)

```

得到的 D 为:

```

  age gender
1   6  Male
2   7  Male
3   8  Male
4   6 Female
5   7 Female
6   8 Female

```

expand.grid() 函数在机器学习中, 特别是交叉检验中为模型选定参数时很实用。

在 R 的实际使用中, 对于很多数据, 通常都是通过数据框的形式从硬盘上导入、导出。在 R 中, 常用的文件读取函数包括 read.table()、read.csv()、read.delim() 等。例如, 对于 Data/Test-Chapter2.csv, 可以通过以下代码将其导入内存, 并保存在数据框 D 中:

```
D <- read.csv('Data/Test-Chapter2.csv')
```

注意, 无论是 Windows 系统还是 Unix 系统, R 中的文件路径永远使用分隔符/。读取 D

后, 获得如下输出:

	Year	Volume	Number
1	2010	3000	307
2	2011	3500	350
3	2012	4000	480
4	2013	4500	550
5	2014	5000	700
6	2015	5500	800
7	2016	6000	1000

对于 csv 文件 (comma separated values file, 即以逗号作为分隔符的文件), 在 R 中最常用的读取函数就是 `read.csv()`; 而 `read.table()` 函数比 `read.csv()` 函数更加通用一些。在 `read.csv()` 函数中, 常用的控制参数有:

- `header`, 一个逻辑型值, 表示文件是否有文件头;
- `sep`, 一个字符变量, 表示文件的分隔符。默认值是 `","`。对于一般的 csv 文件, 不需设置。对于其他文件, 如由制表符分隔的文件, 可将 `sep` 设为 `"\t"`。

同样, 我们也可以将 R 中的数据写到文件中。与前面导入函数对应的导出函数是 `write.table()` 和 `write.csv()` (注意没有 `write.delim()` 函数)。利用 `write.table()` 和 `write.csv()` 函数, 我们可以将数据框写到文本文件 `Data/Test-Chapter2-rewrite.csv` 中:

```
write.csv(D, 'Data/Test-Chapter2-rewrite.csv')
```

更多函数的用法可以直接使用 `help` 函数来获取。

2.4.5 列表

R 的列表 (list) 是一个由对象的有序集合构成的对象。列表中包含的对象又称为它的分量 (component)。列表中的分量可以为任何类型, 包括列表。下面给出列表的几个具体例子:

```
x <- list(1:4, 'b', c('ca', 'cb', 'cc'), c(FALSE, TRUE, TRUE), c(1.9, -1.7))
y <- list(c1=1:4, c2=c('ca', 'cb', 'cc'), c3=c(FALSE, TRUE, TRUE))
```

这里的两个列表 `x` 和 `y` 中都包含了多种不同类型的数据。列表在 R 中使用广泛, 后面讨论的很多机器学习算法返回的模型就是以列表形式保存的。注意, 与向量不同的是, 在 R 中我们需要调用 `list()` 函数而不是 `c()` 函数来构建列表。构建完列表后, 可以使用 `length()` 函数得到其中分量的数目, 使用 `is.list()` 函数来判定一个对象是否是列表对象, 使用 `c()` 函数合并多个列表。

对于列表, 我们可以同时使用 `"[[]]"` 和 `"[]"` 来操作列表。使用 `"[[]]"` 可以访问和修改列表中的单个成员; 使用 `"[]"` 可以得到列表的一个子列表, 其结果仍然是列表。对于列表要特别注意 `"[[]]"` 和 `"[]"` 的区别。下面用具体的例子加以详细说明, 假设 `x` 是上面定义的列表。

```
> x1 <- x[[1]]
```

```
> x1
```

```

[[1]]
[1] 1 2 3 4

> is.list(x1)
[1] TRUE
> x2 <- x[1:2]
> x2
[[1]]
[1] 1 2 3 4

[[2]]
[1] "b"

> is.list(x2)
[1] TRUE
> x3 <- x[[1]]
> x3
[1] 1 2 3 4
> is.list(x3)
[1] FALSE

```

这里 $x[1]$ 和 $x[1:2]$ 都是列表，而 $x[[1]]$ 则是 x 中的第一个分量，不是列表类型。

我们可以进一步修改列表 x 中的数据：

```

x[[2]] <- c('b', 'c')
x[[3]][1] <- 'ca_new'

```

这里我们将其第二个成员改为一个向量，其第三个成员是向量，我们将该向量中的第一个元素改为 'ca_new'。修改之后的 x 如下：

```

> x
[[1]]
[1] 1 2 3 4

[[2]]
[1] "b" "c"

[[3]]
[1] "ca_new" "cb" "cc"

[[4]]
[1] FALSE TRUE TRUE

[[5]]
[1] 1.9 -1.7

```

列表一个很方便使用的特点是，可以给列表中的每个成员赋予一个名字，这样就可以使用名字存取数据了。使用 `names()` 函数可以得到列表中每个分量对应的名字。同时也可以使

用 `names()` 函数来修改各个分量的名字。在前面的例子中，列表 `x` 的各个分量没有名字：

```
> names(x)
NULL
```

使用如下语句可以给各分量命名：

```
> names(x) <- c('c_int', 'c_ch1', 'c_ch2', 'c_logical', 'c_numeric')
> names(x)
[1] "c_int"      "c_ch1"      "c_ch2"      "c_logical"  "c_numeric"
```

在列表中，可以用 `"[[]]"` 或者 `"$"` 来存取和修改分量，如下面的例子所示：

```
> x$c_int
[1] 1 2 3 4
> x[["c_int"]]
[1] 1 2 3 4
> x$c_i
[1] 1 2 3 4
```

可以看出 `x$c_int` 等价于 `x[["c_int"]]`。注意 `x$c_int` 和 `x$c_i` 都对应 `x` 中的第一个成员，这是因为列表中分量名字可以简写，只要能很好地区分各个分量就行。也可以使用 `"[]"` 和分量的名字来获取和修改子序列：

```
> x[c('c_int', 'c_numeric')]
$c_int
[1] 1 2 3 4

$c_numeric
[1] 1.9 -1.7
```

当成员的名字保存在另一个变量中时，使用 `"[[]]"` 和成员名字的做法特别有效。例如，可以用如下语句将 `x$c_int` 和 `x$c_numeric` 中的所有元素都乘以 2：

```
name_list <- c('c_int', 'c_numeric')
for (n in name_list)
  x[[n]] <- 2 * x[[n]]
```

当然，也可以在使用 `list()` 函数构建列表时直接给每个分量赋予相应的名字：

```
> y <- list(c1=1:4, c2=c('ca', 'cb', 'cc'), c3=c(FALSE, TRUE, TRUE))
> y
$c1
[1] 1 2 3 4

$c2
[1] "ca" "cb" "cc"

$c3
[1] FALSE TRUE TRUE
```

此外，列表也是可以扩充的。我们可以往上面的 `y` 中再增加一个或者多个分量：

```

> y[[4]] <- c(4,1)
> y[["c_num"]] <- c(1.9, -1.7)
> y
$c1
[1] 1 2 3 4

$c2
[1] "ca" "cb" "cc"

$c3
[1] FALSE TRUE TRUE

[[4]]
[1] 4 1

$c_num
[1] 1.9 -1.7

```

当我们使用 `y[[4]]` 增加第四个分量时没有给定相应的名字，所以其名字为空。这里可以使用 `names()` 函数来检查 `y` 中各分量的名字：

```

> names(y)
[1] "c1" "c2" "c3" "" "c_num"

```

函数 `c()` 也适用于列表，其功能是将多个列表合并成一个列表。下面是一个简单的例子：

```

> L1 <- list(c1=c(1,4), c2=c('aa', 'dd'))
> L2 <- list(c3=c(TRUE, FALSE), c4=c(4.3, 1.5))
> LC <- c(L1, L2)

```

```

> L1
$c1
[1] 1 4

```

```

$c2
[1] "aa" "dd"

```

```

> L2
$c3
[1] TRUE FALSE

```

```

$c4
[1] 4.3 1.5

```

```

> LC
$c1
[1] 1 4

```

```

$c2
[1] "aa" "dd"

```

```
$c3
[1] TRUE FALSE

$c4
[1] 4.3 1.5
```

2.4.6 下标系统

利用 R 中的下标系统，可以方便地访问 R 中很多对象的元素。在 R 中，下标可以是数值型的或者逻辑型的。前面已经讲解了一些下标使用的例子，这里再深入讨论一下。

首先，在 R 中使用下标必须使用中括号，而小括号则是用来指定函数参数的。例如，`x[1]` 表示 `x` 中的第一个元素，而 `x(1)` 则指调用函数 `x`，且指定输入参数为 1。其次，下标值从 1 开始。再次，下标可以根据元素类型分为数值型和逻辑型，也可以根据维度分为一维、二维等。

数值型的下标是最简单的下标。对于向量 `x=1:10`，可以使用 `x[3]` 访问或者改变 `x` 中的第 3 个元素：

```
x[3] <- 0
```

下标也可以是一个数值型的向量。下面的代码将 `x` 中的第 1 个和第 3 个元素置为 0：

```
x[c(1,3)] <- 0
```

当然也可以使用序列来指定 `x` 中的多个元素：

```
x[3:6] <- 0
```

上述语句表示将 `x` 中的第 3 个~第 6 个元素置为 0。

此外，负号也可以用在下标中，表示排除对应下标的意思。例如，`x[-3]` 表示除第 3 个元素外的所有元素，`x[-c(1,3)]` 表示除第 1 个和第 3 个元素之外的所有元素，`x[-(3:6)]` 表示除第 3~6 个元素之外的所有元素。下面的代码将 `x` 中的第 1 个和第 3 个元素置为 0，其他元素置为 1：

```
x <- 1:10
x[c(1,3)] <- 0
x[-c(1,3)] <- 1
```

执行完代码之后 `x` 为：

```
> x
[1] 0 1 0 1 1 1 1 1 1 1
```

第二类下标是逻辑型下标。使用逻辑型下标时，下标是逻辑型值，一般是某些判定条件。我们仍以向量为例进行说明。`x[x<5] <- 0` 表示将 `x` 中小于 5 的元素全部置为 0，`x[x%%2==0] = 0` 表示将 `x` 中的偶数全部置为 0。

对于矩阵或者数据框 `M`，可以使用二维的下标系统来访问，如 `M[i, j]` 访问 `M` 中第 `i` 行第 `j` 列的元素。此外，还可以通过下标系统来访问一行或者一列，如 `M[i,]` 访问第 `i` 行，`M[,j]`

访问第 j 列。

可以使用 `data()` 函数来载入 R 或者 R 中软件包提供的数据库。例如，可以用 `data(iris)` 载入 R 中提供的 `iris` 数据，之后得到一个变量名和数据名相同的数据框。下面的代码载入了 `iris` 数据并显示了前面几行：

```
data(iris)
head(iris)
```

对应的输出如下：

```
> data(iris)
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4          0.2  setosa
2          4.9         3.0          1.4          0.2  setosa
3          4.7         3.2          1.3          0.2  setosa
4          4.6         3.1          1.5          0.2  setosa
5          5.0         3.6          1.4          0.2  setosa
6          5.4         3.9          1.7          0.4  setosa
```

可以看出，`iris` 数据框中含有 5 个列：`Sepal.Length`、`Sepal.Width`、`Petal.Length`、`Petal.Width` 和 `Species`。在 R 中可以使用列名来访问和改变数据框中的元素。具体来说，我们可以使用 `iris$Sepal.Width` 来访问 `Sepal.Width` 列。利用列名，还可以进一步访问其中的元素，例如：

- 访问 `iris` 中 `Sepal.Width` 列中的第 2 个元素，即 `iris$Sepal.Width[2]`、`iris[2,2]` 或 `iris[2, 'Sepal.Width']`；
- 访问 `iris` 中 `Sepal.Width` 列中的第 1~10 个元素，即 `iris$Sepal.Width[1:10]`、`iris[1:10,2]` 或 `iris[1:10, 'Sepal.Width']`。

2.5 公式对象和 apply 函数

本节介绍 R 中的一些使用技巧，包括 R 中的公式对象及 `apply` 函数族。

在 R 中，利用公式对象（formula object），可以描述建模时各个变量之间的关系。假设要构建一个线性回归模型，我们希望用变量 x_1 、 x_2 、 x_3 来预测变量 y 。从数学上讲，这几个变量之间的关系可以使用下面的公式来描述：

$$y = a \times x_1 + b \times x_2 + c \times x_3 + d$$

在 R 中，可以利用公式 $y \sim x_1 + x_2 + x_3$ 来描述它们之间的关系。可以看出，公式中加号的含义与 R 中加号通常的含义不同。

公式的典型形式是 $y \sim \text{model}$ ，其中 y 是响应变量， model 是一些元素项的集合，而 \sim 表示 y 是 model 的一个函数。特别地，公式 $y \sim$ 表示 y 是因变量，而其他变量是自变量。

在下面的代码中，我们使用 `mtcars` 来构建一个简单的线性回归模型，R 提供了该数据集，

我们只需要使用 `data(mtcars)` 就可以载入数据。

```
data(mtcars)
formula_demo <- as.formula(mpg ~ cyl + disp + hp + gear)
class(formula_demo)
M_lm <- lm(formula_demo, mtcars)
class(M_lm)
print(M_lm)
summary(M_lm)
```

这里想用 `cyl`、`disp`、`hp`、`gear` 这 4 个变量来预测 `mpg` 变量，因此，首先构建了一个 `formula_demo` 公式对象。我们使用 `class(formula_demo)` 检查 `formula_demo` 的类别，其对应的输出为：

```
> class(formula_demo)
[1] "formula"
```

可以看到我们创建了一个公式对象。接下来，我们使用该公式和数据框 `mtcars` 构建一个线性回归模型 `M_lm`，并使用 `class(M_lm)` 来检查 `M_lm` 的类别，最后使用 `print` 和 `summary` 函数来输出 `M_lm` 的相关信息，其输出如下：

```
> class(M_lm)
[1] "lm"
> print(M_lm)
```

```
Call:
lm(formula = formula_demo, data = mtcars)
```

Coefficients:

(Intercept)	cyl	disp	hp	gear
27.4234	-0.8662	-0.0119	-0.0305	1.4231

```
> summary(M_lm)
```

Call:

```
lm(formula = formula_demo, data = mtcars)
```

Residuals:

Min	1Q	Median	3Q	Max
-4.3716	-2.3319	-0.8279	1.3156	7.0782

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	27.42342	6.30108	4.352	0.000173 ***
cyl	-0.86624	0.84941	-1.020	0.316869
disp	-0.01190	0.01190	-1.000	0.325996
hp	-0.03050	0.01982	-1.539	0.135498
gear	1.42306	1.21053	1.176	0.250030

```
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 3.035 on 27 degrees of freedom
```

```
Multiple R-squared: 0.7792, Adjusted R-squared: 0.7465
```

```
F-statistic: 23.82 on 4 and 27 DF, p-value: 1.606e-08
```

R 中的统计模型通常返回一个类名与函数名相同的对象。例如，在上面的例子中，`lm` 函数返回一个 `lm` 的对象。我们可以使用 `print` 和 `summary` 函数来打印 `lm` 对象 `M_lm` 的相关信息。一般来讲，对于 R 中的模型，`print` 函数通常输出简短的汇总信息，而 `summary` 通常给出更为详细的汇总信息。注意，也可以直接使用 `print` 和 `summary` 函数来处理其他对象。例如，我们可以使用 `glmnet` 函数得到一个 `glmnet` 对象，也可以使用这两个函数来处理。与 C++ 中的函数类似，这些函数称为泛型 (generic)。

在 R 中，可以使用 `for`、`while` 等语句实现循环。由于 R 中的基本类型就是向量，因此很多情况下循环语句可以避免。此外，在 R 中可以使用向量化来更高效地实现循环。例如，对两个同样长度的向量 `u` 和 `v` 求和，可以直接使用

```
w <- u + v
```

而不必使用循环

```
w <- rep(0, length(u))
for (i in 1:length(u)) w[i] = u[i] + v[i]
```

特别地，我们介绍一下 `apply` 函数。`apply` 函数作用于矩阵和数据框的行或者列。假设 `M` 是一个矩阵或者数据框，`func` 表示一个函数，`apply` 的常用方法包括：

- `apply(M, 1, func, args)`——对于 `M` 的每行执行函数 `func`，且函数的参数为 `args`；
- `apply(M, 2, func, args)`——对于 `M` 的每列执行函数 `func`，且函数的参数为 `args`。

例如，`apply(M, 1, mean)` 计算每行的平均值，`apply(M, 2, sum)` 计算每列的和。

在 R 中，还有其他的 `apply` 函数，如 `tapply`、`sapply` 和 `lapply`，在这里就不一一介绍了。

2.6 R 软件包

所有的 R 函数和数据集都存放在包 (package) 中，安装相应的包之后才能使用。本章之所以在前面的讨论中跳过了这一概念，是因为 R 的安装环境中已经集成了数十个常用的包。例如，在 2.3.2 节介绍的基本函数中，`ls()`、`rm()`、`paste()`、`cat()`、`c()` 来自 `base` 包，`help()` 来自 `utils` 包，`plot()` 来自 `graphics` 包。

R 软件包分为标准包和贡献包两类。标准包是 R 源代码的一部分，R 环境安装后即可直接使用；贡献包则由全世界的 R 语言爱好者自行开发，读者可以根据自身需求选择性下载使

用^①。截至 2016 年 4 月, R 标准包包括 base、compiler、datasets、grDevices、graphics、grid、methods、parallel、splines、stats、stats4、tcltk、tools、utils 等 14 个; R 贡献包超过 8000 个, 其中 KernSmooth、MASS、Matrix、boot、class、cluster、codetools、foreign、lattice、mgcv、nlme、nnet、rpart、spatial、survival 这 15 个备受推崇的贡献包也已集成到了 R 安装环境中。本书将介绍机器学习中常用的 R 软件包。

2.6.1 软件包的安装

调用 search() 函数可以看到, 在上面提到的 29 个随 R 环境一起安装的软件包中, 有 7 个随着 R 的运行直接进入了内存:

```
[1] ".GlobalEnv"      "tools:rstudio"    "package:stats"
[4] "package:graphics" "package:grDevices" "package:utils"
[7] "package:datasets" "package:methods"  "Autoloads"
[10] "package:base"
```

其他已安装但未调入内存的包可以通过 library() 函数加载。例如, 如果想加载 compiler 包, 则可调用 library('compiler') 或 library(compiler)。如果直接调用 library(), 则可以看到当前系统中已安装的所有软件包。

我们可以使用函数 install.packages 来安装相应的包。下面的代码安装了 glmnet 包以及它所依赖的包:

```
install.packages('glmnet', dependencies=T)
```

也可以在 RStudio 提供的 GUI 界面上安装。单击 Tools, 选择 Install Packages..., RStudio 会弹出如图 2-3 所示的界面。在 Packages 中输入要安装的包的名字, 单击 Install 按钮就可以了。

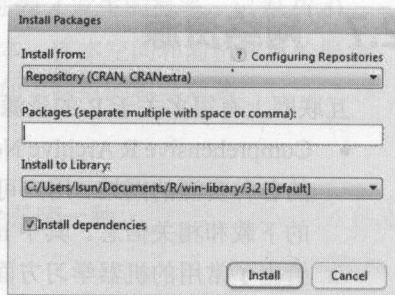


图 2-3 RStudio 中安装包的界面

在运行大量的 R 的代码时, 当我们要调用某个包时可能不知道其是否已经安装。一种简单的办法是先检查该包是否已经安装, 如果没有安装, 就先安装再调用, 如果已经安装, 则直接调用。下面的代码显示了如何先检查 glmnet 的安装情况并调用该包。

```
glmnet.installed <- 'glmnet' %in% rownames(installed.packages())
if (glmnet.installed) {
  print("the glmnet package is already installed, let's load it...")
}else {
  print("let's install the glmnet package first...")
  install.packages('glmnet', dependencies=T)
}
library('glmnet')
```

① 下载地址 <https://cran.rstudio.com/web/packages/>。

如果不能自动找到所需的安装包,则可以手工下载软件包,然后通过 RStudio 的 Tools→Install Packages...菜单命令或 R GUI 的“程序包”→“从本地 zip 文件安装程序包”菜单命令安装。

对于已经安装过的包,可以定期调用 `update.packages()` 加以更新。

2.6.2 软件包的使用

软件包安装后,我们即可使用其中的函数。以 `utils` 包为例,下列调用将列出该包中的函数名称和简要功能说明:

```
library(help = utils)
```

需要注意的是,这一列表对同类函数进行了合并,因此并不完整。例如, `tail()` 函数也归类在 `head()` 函数下面。如果需要详细了解某个函数,可参考 2.3.2 节中 `help()` 函数的用法。

2.6.3 软件包的开发

参考文献[20]详细介绍了如何开发软件包和撰写 R 帮助文件,感兴趣的读者可以到 <https://cran.rstudio.com/manuals.html#R-exts> 免费下载。

2.7 网络资源

互联网上有很多关于 R 的资源,在这里我们简要列出一些最主要的网络资源。

- Comprehensive R Archive Network (CRAN, <https://cran.r-project.org/>) 是一个关于 R 的各种资源的网站。用户可以从这里下载 R 的安装程序。此外,该网站包含了各种包的下载和相关信息。其中 <https://cran.r-project.org/web/views/MachineLearning.html> 上给出了常用的机器学习方面的包。
- <http://www.r-tutor.com/r-introduction> 是一个很好的关于 R 入门的网站。
- <http://www.r-bloggers.com/> 则是一个关于 R 的最新进展的网站,上面常常有一些介绍 R 的很多包用法文章。

关于 R 的入门,我们还推荐 Emmanuel Paradis 所著的《R for Beginners》。该手册有中译本,感兴趣的读者可以参考阅读。

第3章 数学基础

本章介绍机器学习算法中经常用到的概率、统计及矩阵计算方面的基础知识，它们是后继章节讨论具体机器学习算法的基础。数学基础较好的读者可以直接跳过本章。

3.1 概率

3.1.1 基本概念

我们在日常生活中遇到的事件可分为确定性事件和随机事件两类。对于确定性事件，我们可以在一定的条件下预先知道其是否会发生，如“明天早上9点开会”。对于随机事件，我们无法预先知道其是否会发生，但可以研究其发生的可能性，如“从1~10中随机取出的两个数为奇一偶”。本章把随机事件 A 发生的可能性大小称为 A 发生的概率，记为 $P(A)$ 。

在给出概率的数学定义之前，首先引入随机试验、样本、样本空间和随机变量的概念。

(1) 随机试验。为了研究某个随机事件而进行的具有如下特点的科学观测：

- 在相同条件下可重复进行；
- 结果的范围明确可知，且结果不止一种；
- 每次试验恰好获得一种结果，但试验之前不知道会是哪一种。

(2) 样本。随机试验的每种可能结果，称为一个样本。

(3) 样本空间。所有样本组成的集合，称为样本空间。

(4) 随机变量。设随机试验的样本空间为 Ω ，若存在函数 $X()$ ，使得 $\forall w \in \Omega$ ，都存在一个对应的实数 $X(w)$ ，则称 $X()$ 为随机变量。不难看出，随机事件可以用随机变量来表示。以前面的“从1~10中随机取出的两个数为奇一偶”事件为例，若把从1~10中随机取出的两数之和记为 X ，则该事件可等价描述为“ X 为奇数”。

若随机变量 X 的取值范围是可列的，则称 X 为离散型随机变量。这里讲的“可列”指可以按确定的顺序线性地列出，即对于任何一个取值，可以明确指出它的前一个取值、后一个取值分别是什么。具体包括如下两种情形之一：

- 取值范围有限，如 $\{3, 9, 5, 12, 5\}$ ；
- 取值范围无限，如正整数倒数的集合 $\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots\right\}$ 。注意，实数集之类的集合是不可列的，证明从略。

对于离散型随机变量 X , 假设 X 的所有可能取值集合为 $\Omega = \{x_1, x_2, x_3, \dots\}$, 各取值出现的概率分别为 $\{p_1, p_2, p_3, \dots\}$, 而 X 所描述的随机事件 A 对应的样本集合为 $S(A) = \{x_{A_1}, x_{A_2}, x_{A_3}, \dots\} \subseteq \Omega$, 则 A 发生的概率为:

$$P(A) = \sum_{x_{A_i} \in S(A)} p_{A_i} \quad (3-1)$$

特别地, 当取值范围有限且各取值出现的概率均等时, 随机事件 A 发生的概率为:

$$P(A) = \frac{|S(A)|}{|\Omega|} \quad (3-2)$$

其中 $||$ 表示集合中元素的数量。式 (3-2) 就是著名的古典概率定义。

对于随机变量 X , 若存在非负可积的函数 $p(x)$, 使得

$$\forall x \in \mathbb{R}, P(X \leq x) = \int_{-\infty}^x p(t) dt \quad (3-3)$$

则称 X 为连续型随机变量, 称函数 $p(x)$ 为 X 的概率密度函数。这种情况下, 假设随机事件 A 对应的样本区间为 $R(A)$, 则 A 发生的概率为:

$$P(A) = \int_{t \in R(A)} p(t) dt \quad (3-4)$$

注意, $p(x)$ 不一定是连续函数。

此外, 对于一些既非离散型也非连续型的随机变量, 可以分段考虑。

3.1.2 基本公式

概率具有如下基本性质:

$$\begin{cases} P(\Phi) = 0 \\ P(\Omega) = 1 \\ \forall A, 0 \leq P(A) \leq 1 \end{cases} \quad (3-5)$$

其中 Φ 对应于空集, 表示不可能事件; 样本空间 Ω 对应于全集, 表示必然事件。注意, 式 (3-5) 中第一个等式仅表明不可能事件发生的概率为 0; 事实上, 概率为 0 的随机事件未必是不可能事件, 见 3.1.4 节。

对于随机事件 A, B , 如果用 $A \cup B$ 表示 A, B 中至少有一个发生, 用 $A \cap B$ 表示 A, B 同时发生, 则有

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \quad (3-6)$$

如果 A 和 B 互斥, 则 $A \cap B = \Phi$, 代入式 (3-6) 得

$$P(A \cup B) = P(A) + P(B) \quad (3-7)$$

一般来说, 若 A_1, A_2, A_3, \dots 为可列无穷个互斥事件, 则有

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i) \quad (3-8)$$

若随机事件 A 发生一定会导致随机事件 B 发生, 则有

$$P(B-A) = P(B) - P(A) \quad (3-9)$$

这里, $P(B-A)$ 表示事件 B 发生而事件 A 不发生的概率。

当 $P(B) > 0$ 时, 称

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (3-10)$$

为在已知随机事件 B 发生的前提下, 随机事件 A 发生的条件概率。写成乘法形式为

$$P(A \cap B) = P(A|B)P(B) \quad (3-11)$$

如果存在一组互斥事件 $A_1, A_2, A_3, \dots, A_n$, 满足

$$\begin{cases} P(A_i) > 0, i=1, 2, 3, \dots, n \\ \bigcup_{i=1}^n A_i = \Omega \end{cases} \quad (3-12)$$

则对任意随机事件 B , 有

$$P(B) = \sum_{i=1}^n P(B|A_i)P(A_i) \quad (3-13)$$

式 (3-13) 称为全概率公式。若 $P(B) > 0$, 则根据式 (3-10), $\forall i \in \{1, 2, \dots, n\}$ 有

$$P(A_i|B) = \frac{P(A_i \cap B)}{P(B)} = \frac{P(A_i \cap B)}{\sum_{i=1}^n P(B|A_i)P(A_i)} \quad (3-14)$$

根据式 (3-11), 将 $P(A_i \cap B)$ 替换为 $P(B|A_i)P(A_i)$, 可得如下的贝叶斯公式:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{i=1}^n P(B|A_i)P(A_i)} \quad (3-15)$$

如果随机事件 A, B 满足条件

$$P(A \cap B) = P(A)P(B) \quad (3-16)$$

则称事件 A, B 相互独立。对照式 (3-11) 不难看出, 当 $P(B) > 0$ 时, 式 (3-16) 等价于

$$P(A|B) = P(A) \quad (3-17)$$

也就是说, 随机事件 B 是否发生不会影响随机事件 A 发生的概率。

3.1.3 常用分布

1. 常用离散分布

(1) 0-1 分布 (0-1 distribution)。

如果随机变量 X 满足

$$P(X=k)=\begin{cases} p, & \text{如果 } k=1 \\ 1-p, & \text{如果 } k=0 \end{cases} \quad (3-18)$$

这里 $0 \leq p \leq 1$, 则称 X 服从 0-1 分布, 也称为伯努利分布 (Bernoulli distribution)。在实际应用中, 若随机试验仅有两种可能的结果, 则可以定义服从 0-1 分布的随机变量加以描述。在机器学习中, 由于 k 的取值只能为 0 或者 1, 因此我们通常将 $P(X=k)$ 写为如下形式:

$$P(X=k)=p^k(1-p)^{1-k}$$

(2) 几何分布 (geometric distribution)。

如果随机变量 X 满足

$$P(X=k)=p(1-p)^{k-1} \quad (3-19)$$

其中 $k=1, 2, 3, \dots$, $0 \leq p \leq 1$, 则称 X 服从几何分布 $g(p)$, 记为 $X \sim g(p)$ 。在实际应用中, 若多次随机试验相互独立, 则可以定义服从几何分布的随机变量来描述随机事件的“首次出现”。

(3) 二项分布 (binomial distribution)。

如果随机变量 X 满足

$$P(X=k)=\binom{n}{k}p^k(1-p)^{n-k} \quad (3-20)$$

其中 $k=0, 1, 2, \dots, n$, $0 \leq p \leq 1$, $\binom{n}{k}=\frac{n!}{k!(n-k)!}$, 则称 X 服从二项分布 $B(n, p)$, 记为 $X \sim B(n, p)$ 。

在实际应用中, 若多次随机试验相互独立, 则可以定义服从二项分布的随机变量来描述随机事件的出现次数。

2. 常用连续分布

对于连续型随机变量 X , 设其概率密度函数为 $p(x)$ 。

(1) 均匀分布 (uniform distribution)。

假设 $b > a$, 若 $x \in [a, b]$ 时有

$$p(x)=\frac{1}{b-a} \quad (3-21)$$

且 $x \notin [a, b]$ 时 $p(x)=0$, 则称 X 服从均匀分布 $U[a, b]$, 记为 $X \sim U[a, b]$ 。在实际应用中, 若随机试验的结果落在区间 $[a, b]$ 内任意一点的可能性都相等, 则可以定义服从均匀分布的随机变量加以描述。

(2) 指数分布 (exponential distribution)。

设 $\lambda > 0$ 为常数, 若 $\forall x \in [0, +\infty)$,

$$p(x) = \lambda e^{-\lambda x} \quad (3-22)$$

且 $x \in (-\infty, 0)$ 时 $p(x) = 0$, 则称 X 服从指数分布 $E(\lambda)$, 记为 $X \sim E(\lambda)$ 。在实际应用中, 若随机试验涉及元器件使用寿命、系统等待时间等, 则往往可以定义服从指数分布的随机变量加以描述。

(3) 正态分布 (normal distribution)。

若 $\forall x$,

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (3-23)$$

其中 μ 、 σ 均为常数且 $\sigma > 0$, 则称 X 服从正态分布 $N(\mu, \sigma^2)$, 记为 $X \sim N(\mu, \sigma^2)$ 。在实际应用中, 若随机试验受多种独立随机因素的影响, 则往往可以定义服从正态分布的随机变量加以描述。特别地, $\mu=0$ 、 $\sigma=1$ 时的正态分布称为标准正态分布。图 3-1 是标准正态分布 $N(0, 1)$ 的密度函数图像。

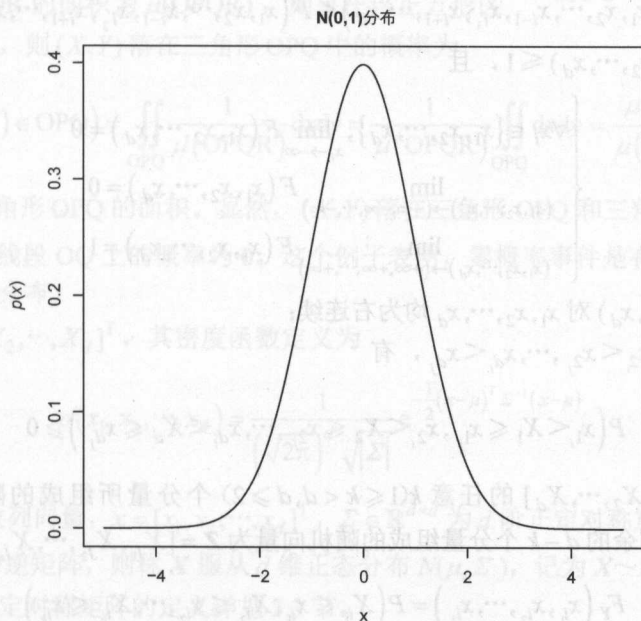


图 3-1 标准正态分布 $N(0, 1)$ 的密度函数图像

3.1.4 随机向量及其分布

1. 随机向量与分布函数

前面讨论了单个随机变量及其常见分布, 但有时用单个随机变量难以描述随机事件, 需

要把多个随机变量作为一个整体进行研究。设随机试验的样本空间为 Ω ，若存在向量 $[X_1, X_2, \dots, X_d] \in \mathbb{R}^d$ ，使得 $\forall w \in \Omega$ ，都存在一组对应的实数 $[X_1(w), X_2(w), \dots, X_d(w)]$ ，则称 $[X_1, X_2, \dots, X_d]$ 为 d 维随机向量。由前面随机变量的定义可知， d 维随机向量是由 d 个随机变量组成的。特别地，若 $\forall x_1, x_2, \dots, x_d \in \mathbb{R}$ ，有

$$P(X_1 \leq x_1, X_2 \leq x_2, \dots, X_d \leq x_d) = P(X_1 \leq x_1)P(X_2 \leq x_2) \cdots P(X_d \leq x_d) \quad (3-24)$$

则称这 d 个随机变量相互独立 ($d \geq 2$)。为了讨论方便，我们通常把 $P(X_1 \leq x_1, X_2 \leq x_2, \dots, X_d \leq x_d)$ 称为随机向量 $\mathbf{X} = [X_1, X_2, \dots, X_d]$ 的分布函数，记为 $F(x_1, x_2, \dots, x_d)$ 。从而，式 (3-24) 可等价写为

$$F(x_1, x_2, \dots, x_d) = F_1(x_1)F_2(x_2) \cdots F_d(x_d) \quad (3-25)$$

这里 $F_1(x_1) = P(X_1 \leq x_1)$, $F_2(x_2) = P(X_2 \leq x_2)$, \dots , $F_d(x_d) = P(X_d \leq x_d)$ 分别是随机变量 X_1, X_2, \dots, X_d 的分布函数。

随机向量的分布函数 $F(x_1, x_2, \dots, x_d)$ 具有如下性质：

- 对任意给定的 $\{x_1, x_2, \dots, x_d\} \setminus \{x_i\} (i \in \{1, 2, \dots, d\})$ ，若 $x_{i_1} < x_{i_2}$ ，则

$$F(x_1, x_2, \dots, x_{i_1}, x_{i_2}, \dots, x_d) \leq F(x_1, x_2, \dots, x_{i_1}, x_{i_2}, x_{i_2}, \dots, x_d) \quad (3-26)$$

- $0 \leq F(x_1, x_2, \dots, x_d) \leq 1$ ，且

$$\begin{cases} \forall x_i \in \{x_1, x_2, \dots, x_d\}, \lim_{x_i \rightarrow -\infty} F(x_1, x_2, \dots, x_d) = 0 \\ \lim_{(x_1, x_2, \dots, x_d) \rightarrow (-\infty, -\infty, \dots, -\infty)} F(x_1, x_2, \dots, x_d) = 0 \\ \lim_{(x_1, x_2, \dots, x_d) \rightarrow (+\infty, +\infty, \dots, +\infty)} F(x_1, x_2, \dots, x_d) = 1 \end{cases} \quad (3-27)$$

- $F(x_1, x_2, \dots, x_d)$ 对 x_1, x_2, \dots, x_d 均为右连续；
- $\forall x_{i_1} < x_{i_2}, x_{j_1} < x_{j_2}, \dots, x_{d_1} < x_{d_2}$ ，有

$$P(x_{i_1} < X_1 \leq x_{i_2}, x_{j_1} < X_2 \leq x_{j_2}, \dots, x_{d_1} < X_d \leq x_{d_2}) \geq 0 \quad (3-28)$$

对于 $\mathbf{X} = [X_1, X_2, \dots, X_d]$ 的任意 $k (1 \leq k < d, d \geq 2)$ 个分量所组成的随机向量 $\mathbf{Y} = [X_{i_1}, X_{i_2}, \dots, X_{i_k}]$ ，设剩余的 $d-k$ 个分量组成的随机向量为 $\mathbf{Z} = [X_{j_1}, X_{j_2}, \dots, X_{j_{d-k}}]$ ，把

$$\begin{aligned} F_Y(x_{i_1}, x_{i_2}, \dots, x_{i_k}) &= P(X_{i_1} \leq x_{i_1}, X_{i_2} \leq x_{i_2}, \dots, X_{i_k} \leq x_{i_k}) \\ &= \lim_{\mathbf{Z} \rightarrow (+\infty, +\infty, \dots, +\infty)} F(x_1, x_2, \dots, x_d) \end{aligned} \quad (3-29)$$

称为 $\mathbf{X} = [X_1, X_2, \dots, X_d]$ 的 k 维边缘分布函数。

与随机变量类似，若随机向量 $\mathbf{X} = [X_1, X_2, \dots, X_d]$ 的取值范围是可列的，则称其为 d 维离散型随机向量；若存在非负可积的函数 $p(x_1, x_2, \dots, x_d)$ ，使得

$$\forall x_1, x_2, \dots, x_n \in \mathbb{R}, F(x_1, x_2, \dots, x_d) = \int_{-\infty}^{x_1} \int_{-\infty}^{x_2} \cdots \int_{-\infty}^{x_d} p(t_1, t_2, \dots, t_d) dt_1 dt_2 \cdots dt_d \quad (3-30)$$

则称 \mathbf{X} 为 d 维连续型随机向量, 称函数 $p(x_1, x_2, \dots, x_d)$ 为 \mathbf{X} 的概率密度函数。

不难看出, 当 $d=1$ 时, 随机向量就退化成为随机变量, 本节给出的分布函数定义与性质同样适用于随机变量。

2. 随机向量的常用分布

(1) 二维均匀分布。

假设 S 为二维平面上的有界区域, 其面积为 $\mu(S)$, 随机向量 (X, Y) 的概率密度函数为

$$p(x, y) = \begin{cases} \frac{1}{\mu(S)}, & (x, y) \in S \\ 0, & (x, y) \notin S \end{cases} \quad (3-31)$$

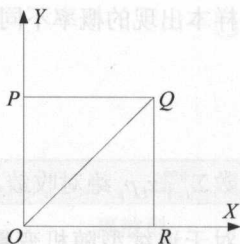


图 3-2 二维均匀分布示例

则称 (X, Y) 服从均匀分布 $U(S)$, 记为 $(X, Y) \sim U(S)$ 。在图 3-2 中, 设正方形 $OPQR$ 的面积为 $\mu(OPQR)$, 如果在该正方形区域内等概率地取点, 则 (X, Y) 落在三角形 OPQ 中的概率为

$$P((X, Y) \in OPQ) = \iint_{OPQ} \frac{1}{\mu(OPQR)} dx dy = \frac{1}{\mu(OPQR)} \iint_{OPQ} dx dy = \frac{\mu(OPQ)}{\mu(OPQR)} \quad (3-32)$$

这里 $\mu(OPQ)$ 为三角形 OPQ 的面积。显然, (X, Y) 落在三角形 OPQ 和三角形 OQR 中的概率都为 0.5, 而落在线段 OQ 上的概率为 0。这个例子表明, 零概率事件是有可能发生的。

(2) d 维正态分布。

若 $\forall \mathbf{X} = [X_1, X_2, \dots, X_d]^T$, 其密度函数定义为

$$p(x_1, x_2, \dots, x_d) = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})} \quad (3-33)$$

其中 $\boldsymbol{\mu} \in \mathbb{R}^d$ 为常数列向量, $\mathbf{x} = [x_1, x_2, \dots, x_d]^T$, $\Sigma \in \mathbb{R}^{d \times d}$ 为 d 阶正定对称矩阵, $|\Sigma|$ 和 Σ^{-1} 分别为 Σ 的行列式与逆矩阵, 则称 \mathbf{X} 服从 d 维正态分布 $N(\boldsymbol{\mu}, \Sigma)$, 记为 $\mathbf{X} \sim N(\boldsymbol{\mu}, \Sigma)$ 。这里的行列式、逆矩阵和正定对称矩阵的定义详见 3.3 节。

特别地, 当 $d=1$ 时, $\Sigma > 0$ 为常数, 式 (3-33) 等价于

$$p(x) = \frac{1}{\sqrt{2\pi}\sqrt{\Sigma}} e^{-\frac{1}{2\Sigma}(x-\mu)^2} \quad (3-34)$$

令 $\sigma = \sqrt{\Sigma}$, 则式 (3-34) 等同于式 (3-23)。

3.1.5 随机变量的数字特征

1. 数学期望

对于离散型随机变量 X , 若样本 $x_1, x_2, \dots, x_n \in \Omega$ 出现的概率都相等, 则可以定义如下的 \bar{x} 来表示这 n 个样本的均值:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (3-35)$$

若各样本出现的概率不同, 用 p_i 表示 x_i 的出现概率 ($i=1, 2, \dots, n$), 则有

$$\bar{x} = \sum_{i=1}^n x_i p_i \quad (3-36)$$

若级数 $\sum_{i=1}^{+\infty} x_i p_i$ 绝对收敛, 则称 $\sum_{i=1}^{+\infty} x_i p_i$ 为 X 的数学期望。

对于连续型随机变量 X , 设其概率密度函数为 $p(x)$, 若 $\int_{-\infty}^{+\infty} xp(x)dx$ 绝对收敛, 则称 $\int_{-\infty}^{+\infty} xp(x)dx$ 为 X 的数学期望。

本书将随机变量 X 的数学期望记为 $E(X)$ 。对于常见的随机变量, 绝对收敛的条件一般都满足, 不需要特别考虑。

数学期望具有如下重要性质:

- 对任意 X_1, X_2, \dots, X_n , $E(X_1 + X_2 + \dots + X_n) = E(X_1) + E(X_2) + \dots + E(X_n)$;
- 对任意常数 a , $E(a) = a$;
- 若 X_1, X_2, \dots, X_n 相互独立, 则 $E(X_1 X_2 \dots X_n) = E(X_1)E(X_2) \dots E(X_n)$ 。

对于 3.1.3 节介绍的常用分布, 其数学期望见表 3-1。

表 3-1 常用分布的数学期望

常用分布	数学期望	备注
0-1 分布	$E(X) = P(X=1)$	离散型
几何分布 $g(p)$	$E(X) = \frac{1}{p}$	离散型
二项分布 $B(n, p)$	$E(X) = np$	离散型
均匀分布 $U[a, b]$	$E(X) = \frac{1}{2}(a+b)$	连续型
指数分布 $E(\lambda)$	$E(X) = \frac{1}{\lambda}$	连续型
正态分布 $N(\mu, \sigma^2)$	$E(X) = \mu$	连续型

2. 方差

数学期望体现了随机变量取值的平均程度，方差则描述了数据相对于数学期望的分散程度。本书将随机变量 X 的方差记为 $\text{var}(X)$ ，则有

$$\text{var}(X) = E\left(\left(X - E(X)\right)^2\right) \quad (3-37)$$

根据数学期望的性质可得

$$\text{var}(X) = E(X^2) - (E(X))^2 \quad (3-38)$$

方差的平方根称为标准差，记为 $\text{std}(X)$ ：

$$\text{std}(X) = \sqrt{\text{var}(X)} \quad (3-39)$$

对于 3.1.3 节介绍的常用分布，其方差见表 3-2。

表 3-2 常用分布的方差

常用分布	方差	备注
0-1 分布	$\text{var}(X) = P(X=1) \cdot P(X=0)$	离散型
几何分布 $g(p)$	$\text{var}(X) = \frac{1-p}{p^2}$	离散型
二项分布 $B(n, p)$	$\text{var}(X) = np(1-p)$	离散型
均匀分布 $U[a, b]$	$\text{var}(X) = \frac{1}{12}(b-a)^2$	连续型
指数分布 $E(\lambda)$	$\text{var}(X) = \frac{1}{\lambda^2}$	连续型
正态分布 $N(\mu, \sigma^2)$	$\text{var}(X) = \sigma^2$	连续型

3. 协方差与相关系数

对于两个随机变量 X 和 Y ，其协方差定义为

$$\text{cov}(X, Y) = E\left(\left(X - E(X)\right)\left(Y - E(Y)\right)\right) \quad (3-40)$$

若 $\text{var}(X) > 0$ 、 $\text{var}(Y) > 0$ 均存在，则 X 与 Y 的相关系数 $cc(X, Y)$ 定义为

$$cc(X, Y) = \frac{\text{cov}(X, Y)}{\sqrt{\text{var}(X)}\sqrt{\text{var}(Y)}} \quad (3-41)$$

当 $cc(X, Y) = 0$ 时称 X 与 Y 不相关。注意， X 与 Y 不相关表明 X 与 Y 之间不存在线性关系，但可能存在其他关系，因此 X 与 Y 不一定相互独立。

相关系数的一个重要性质是 $|cc(X, Y)| \leq 1$ ，即 $\text{cov}(X, Y)^2 \leq \text{var}(X)\text{var}(Y)$ ，证明从略。

4. 偏度

对于随机变量 X ，可以用如下定义的偏度 $\text{skew}(X)$ 来衡量样本数据分布的非对称性：

$$skew(X) = E\left(\left(\frac{X - E(X)}{\sqrt{\text{var}(X)}}\right)^3\right) \quad (3-42)$$

当 $skew(X) < 0$ 时, 样本分布的偏度为负, 此时直线 $X = E(X)$ 左侧的样本数量直观上少于右侧的样本数量; 当 $skew(X) = 0$ 时, 样本分布的偏度为 0, 此时直线 $X = E(X)$ 左侧的样本数量直观上等于右侧的样本数量; 当 $skew(X) > 0$ 时, 样本分布的偏度为正, 此时直线 $X = E(X)$ 左侧的样本数量直观上多于右侧的样本数量。图 3-3 显示了 3 个不同的分布对应的密度函数图像, 直观地显示了不同的偏度值。

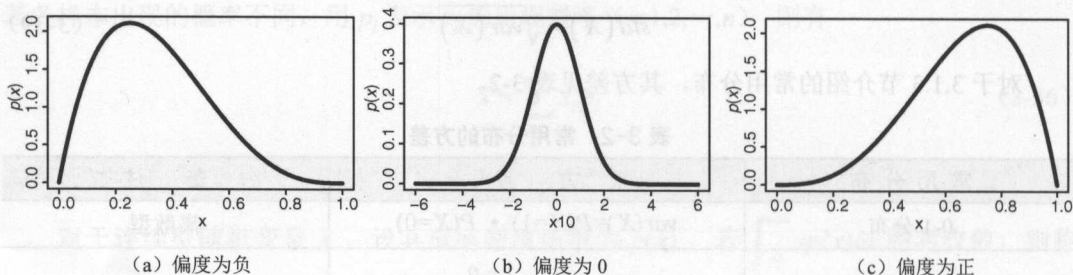


图 3-3 偏度范例

3.1.6 随机向量的数字特征

前面讨论了随机变量的数字特征, 对于实际问题中更常见的随机向量, 相关数字特征的定义是类似的。例如, 对于 n 维随机向量 $\mathbf{X} = [X_1, X_2, \dots, X_d]$, 定义

$$E(\mathbf{X}) = [E(X_1), E(X_2), \dots, E(X_d)] \quad (3-43)$$

为 \mathbf{X} 的数学期望向量, 定义

$$\text{var}(\mathbf{X}) = [\text{var}(X_1), \text{var}(X_2), \dots, \text{var}(X_d)] \quad (3-44)$$

为 \mathbf{X} 的方差向量, 定义

$$\text{std}(\mathbf{X}) = [\sqrt{\text{var}(X_1)}, \sqrt{\text{var}(X_2)}, \dots, \sqrt{\text{var}(X_d)}] \quad (3-45)$$

为 \mathbf{X} 的标准差向量。

定义随机向量的协方差和相关系数时, 必须考虑各分量之间的两两关系。因此, 随机向量 $\mathbf{X} = [X_1, X_2, \dots, X_d]$ 的协方差矩阵 $\text{cov}(\mathbf{X})$ 和相关系数矩阵 $\text{cc}(\mathbf{X})$ 分别由式 (3-46) 和式 (3-47) 定义:

$$\text{cov}(\mathbf{X}) = \begin{bmatrix} \text{cov}(X_1, X_1) & \cdots & \text{cov}(X_1, X_d) \\ \vdots & \ddots & \vdots \\ \text{cov}(X_d, X_1) & \cdots & \text{cov}(X_d, X_d) \end{bmatrix} \quad (3-46)$$

其中 $\text{cov}(X_i, X_j)$ 的定义见式 (3-40),

$$\text{cc}(\mathbf{X}) = \begin{bmatrix} \text{cc}(X_1, X_1) & \cdots & \text{cc}(X_1, X_d) \\ \vdots & \ddots & \vdots \\ \text{cc}(X_d, X_1) & \cdots & \text{cc}(X_d, X_d) \end{bmatrix} \quad (3-47)$$

其中 $\text{cc}(X_i, X_j)$ 的定义见式 (3-41)。

3.2 统计

统计指的是以概率论为基础, 对所得的数据进行收集、整理、分析的过程。在这里, 我们介绍一些在统计学和机器学习中常用的数据特征, 包括均值、方差、标准差、极差、分位数、绝对平均偏差、中位数绝对偏差、四分位极差、偏度、协方差、相关系数等基本概念, 并重点介绍极大似然估计。

3.2.1 常用数据特征

1. 一元数据常用数字特征

在前面的讨论中, 我们讨论了服从某个随机分布的随机变量的一些数字特征, 如期望和方差。在本节中, 我们则从统计的角度来计算数据的数字特征。

在一元数据中, 假设我们有样本

$$x_1, x_2, \dots, x_n$$

其中 n 称为样本容量。在某些简单的问题中, 这 n 个观测值就是所要研究的对象的全体; 更普遍的情况是这 n 个样本是从总体中抽取的样本。在前一种情况中, 我们要利用全部的 n 个样本研究数据中包括的信息; 在后一种情况中, 我们要从有限的 n 个样本中推断总体的信息。

下面我们介绍如何从给定的样本集合中计算数字特征, 包括均值、方差、标准差、中位数、极差、分位数、绝对平均偏差、中位数绝对偏差、四分位极差和偏度等。在机器学习中, 在很多情况下给定的这组样本就是我们所有的数据, 因此从数据中直接计算数字特征对于解数据是非常重要的。

(1) 均值是样本 x_1, x_2, \dots, x_n 的平均值, 记为 \bar{x} :

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (3-48)$$

均值表示了数据的集中位置。

(2) 方差则描述了数据的分散程度, 它度量了一组数据相对于均值的偏离程度。我们把方差记为 s^2 , 其定义为:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (3-49)$$

(3) 方差的开方称为标准差, 我们记为 s :

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} \quad (3-50)$$

注意, 方差的量纲与数据的量纲不一致, 而标准差的量纲与数据的量纲是一致的。

上面讨论的均值、方差、标准差等数字特征适用于来自正态分布的数据。如果数据的分布严重偏向某一边, 或者极端值比较多, 则上述的这些特征就不适用了。在这种情况下, 我们可以计算中位数、分位数、极差等数据特征。在下面的讨论中, 我们假设将数据按从小到大的顺序排好, 并将排好的样本记为

$$x'_1, x'_2, \dots, x'_n$$

(1) 中位数。当 n 为奇数时, 中位数 M 定义为:

$$M = x'_{\frac{n+1}{2}} \quad (3-51)$$

当 n 为偶数时, 中位数 M 定义为:

$$M = \frac{x'_{\frac{n}{2}} + x'_{\frac{n}{2}+1}}{2} \quad (3-52)$$

与均值相比, 中位数基本上不受极端值的影响。此外, 如果数据的分布对称, 中位数与均值会比较接近。但是当数据的分布偏向某一边时, 均值和中位数的差异会比较大。在实际的数据处理中, 中位数是一个非常重要的统计量。

(2) 极差 (range)。对于样本数据 x_1, x_2, \dots, x_n , 极差的定义为:

$$R = \max(x_i) - \min(x_i) \quad (3-53)$$

(3) 分位数。对于 $0 \leq p < 1$, 数据 x_1, x_2, \dots, x_n 的 p 分位数为:

$$M_p = \begin{cases} x'_{[np]+1}, & \text{如果 } np \text{ 不是整数} \\ \frac{x'_{np} + x'_{np+1}}{2}, & \text{如果 } np \text{ 是整数} \end{cases} \quad (3-54)$$

这里 $[np]$ 表示 np 的整数部分。当 $p=1$ 时, 我们定义

$$M_1 = x'_n \quad (3-55)$$

p 分位数又称为 $100p$ 百分位数。常用的 p 分位数包括 0.25 分位数、0.5 分位数和 0.75 分位数。其中 0.5 分位数就是中位数。0.25 分位数也称为下四分位数, 记为:

$$Q_1 = M_{0.25} \quad (3-56)$$

0.75 分位数也称为上四分位数, 记为:

$$Q_3 = M_{0.75} \quad (3-57)$$

我们注意到均值的计算容易受到离群数据或者极端数据的影响。在计算方差时, 我们也需要用到均值。事实上, 方差比均值对极端数据更加敏感。因为对于每个点 x_i , 首先要计算 $x_i - \bar{x}$ 再平方。在平方的过程中, 极端数据的影响会进一步放大。为了降低极端数据的影响, 我们引入如下的统计量来衡量数据散布区间的大小。

(1) 绝对平均偏差 (absolute average deviation, AAD):

$$AAD = \frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}| \quad (3-58)$$

(2) 中位数绝对偏差 (median absolute deviation, MAD):

$$MAD = \text{median}\{|x_1 - \bar{x}|, |x_2 - \bar{x}|, \dots, |x_n - \bar{x}|\} \quad (3-59)$$

(3) 四分位极差 (inter-quartile range, IQR):

$$IQR(x) = M_{0.75} - M_{0.25} = Q_3 - Q_1 \quad (3-60)$$

(4) 偏度。对于样本数据 x_1, x_2, \dots, x_n , 偏度的计算公式是:

$$g = \frac{n}{(n-1)(n-2)s^3} \sum_{i=1}^n (x_i - \bar{x})^3 \quad (3-61)$$

事实上, 这里讨论的很多样本数字特征是相应的总体数字特征的矩估计。当样本容量 n 足够大, 且总体数字特征存在时, 样本数字特征趋近于待估参数的真实值。

2. 多元数据常用数字特征

在实际问题中, 更多的情况下我们碰到的都是多元数据。在多元数据处理中, 除了分析各个分量的数字特征外, 我们更关心变量之间的相互关系。

假设我们有 n 个 d 维向量组成的样本:

$$\mathbf{x}_1 = [x_{11}, x_{12}, \dots, x_{1d}]^T, \mathbf{x}_2 = [x_{21}, x_{22}, \dots, x_{2d}]^T, \dots, \mathbf{x}_n = [x_{n1}, x_{n2}, \dots, x_{nd}]^T \quad (3-62)$$

与一元数据分析类似, 我们可以定义每个变量的均值和方差。对于第 j ($j=1, 2, \dots, d$) 个变量, 其均值 \bar{x}_j 和方差 s_{jj} 定义为

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}, s_{jj} = \frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2 \quad (3-63)$$

注意, 这里我们使用 s_{jj} 来标记第 j 个变量的方差, 因为我们要考虑不同变量之间的协方差, 例如, 使用 s_{jk} 来标记第 j 个变量和第 k 个变量的协方差。根据该样本集合, 我们定义第 j 个变量和第 k 个变量的协方差为

$$s_{jk} = \frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) \quad (3-64)$$

我们可以根据样本数据定义均值向量, 记为 \bar{x} :

$$\bar{x} = [\bar{x}_1, \bar{x}_2, \dots, \bar{x}_d]^T = \left[\frac{1}{n} \sum_{i=1}^n x_{i1}, \frac{1}{n} \sum_{i=1}^n x_{i2}, \dots, \frac{1}{n} \sum_{i=1}^n x_{id} \right]^T \quad (3-65)$$

相应地, 我们定义 S 为样本观测数据的协方差矩阵:

$$S = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1d} \\ s_{21} & s_{22} & \cdots & s_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ s_{d1} & s_{d2} & \cdots & s_{dd} \end{bmatrix} \quad (3-66)$$

如果表示成矩阵的形式, 我们有

$$S = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})^T \quad (3-67)$$

由协方差的定义可知矩阵 S 是对称矩阵, 即

$$S = S^T \quad (3-68)$$

在协方差的基础上, 还可以根据样本数据计算相关系数。可以计算第 j 个变量和第 k 个变量的相关系数如下:

$$r_{jk} = \frac{s_{jk}}{\sqrt{s_{jj}} \sqrt{s_{kk}}} \quad (3-69)$$

相关系数衡量了两个变量之间存在线性关系的程度。当 $r_{jk} = \pm 1$ 时, 这里第 j 个和第 k 个变量之间存在线性关系。类似地, 也可以将相关系数表示成矩阵的形式, 称为相关矩阵:

$$R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1d} \\ r_{21} & r_{22} & \cdots & r_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ r_{d1} & r_{d2} & \cdots & r_{dd} \end{bmatrix} \quad (3-70)$$

相关矩阵是多元观测数据最重要的数字特征之一。在进行多元数据探索时, 相关矩阵是我们首先要计算的若干关键指标之一。

3.2.2 参数估计

前面讨论的数字特征是基于整个样本空间 Ω 定义的。在实际应用中, 我们往往只能选取部分研究对象进行试验, 试验的结果是 Ω 的真子集。本章把全体研究对象所组成的集合称为

母体,把所选取的部分研究对象所组成的集合称为子样,把选取子样的过程称为抽样。为了使子样能较好地反映母体特性,我们要求抽样过程具备如下性质。

- 代表性:子样随机向量的每个分量与母体随机向量具有相同的分布。
- 独立性:子样随机向量的各分量相互独立。

本节讨论在母体概率分布类型已知的前提下,通过分析子样来估计母体概率分布中的未知参数的方法。参数估计的方法主要有点估计、区间估计两类,前者可以得到明确的参数近似值,后者则可以指明某个区间以多大概率包含所求参数。在点估计中,常用的方法有矩估计和极大似然估计。在这里我们介绍极大似然估计(maximum likelihood estimation)。

对于连续型随机变量 X , 假设其密度函数为 $p(x, \theta)$, 其中 θ 为待估计的参数。若 $\{X_1, X_2, \dots, X_n\}$ 是来自母体的一个子样, 则称

$$L(X, \theta) = \prod_{i=1}^n p(x_i, \theta) \quad (3-71)$$

为该子样的似然函数(likelihood function)。

对于离散型随机变量 X , 假设对于任意 i , X 取值 x_i 的概率为 $P(X = x_i, \theta)$, 其中 θ 为待估计的参数。若 $\{X_1, X_2, \dots, X_n\}$ 是来自母体的一个子样, 则称

$$L(X, \theta) = \prod_{i=1}^n P(X = x_i, \theta) \quad (3-72)$$

为该子样的似然函数。

极大似然估计法的基本思想是通过最大化似然函数来求出参数 θ 的估计量。下面通过一个例子加以说明。

例 3-1 假设 $\{X_1, X_2, \dots, X_n\}$ 是母体的一个子样, 求概率密度函数

$$p(x, \theta) = \begin{cases} (\theta+1)x^\theta, & x \in (0, 1) \\ 0, & x \notin (0, 1) \end{cases} \quad (3-73)$$

的极大似然估计量。这里 $\theta > -1$ 是未知参数。

解 似然函数为

$$L(X, \theta) = \prod_{i=1}^n (\theta+1)x_i^\theta = (\theta+1)^n \left(\prod_{i=1}^n x_i \right)^\theta \quad (3-74)$$

两边取对数, 有

$$\ln L(X, \theta) = n \ln(\theta+1) + \theta \cdot \ln \left(\prod_{i=1}^n x_i \right) \quad (3-75)$$

两边对 θ 求导, 令 $\frac{d \ln L(X, \theta)}{d \theta} = 0$, 得

$$\frac{n}{\hat{\theta}+1} + \ln \left(\prod_{i=1}^n x_i \right) = 0 \quad (3-76)$$

于是 θ 的极大似然估计量为

$$\hat{\theta} = -\frac{n}{\ln \left(\prod_{i=1}^n x_i \right)} - 1 = -\frac{n}{\sum_{i=1}^n \ln x_i} - 1 \quad (3-77)$$

3.3 矩阵

在机器学习中, 矩阵是一个非常常用的工具。因为在很多实际问题中, 数据可以很容易地用矩阵来表示。当我们用矩阵来表示数据时, 数据的很多操作可以直接用矩阵的相应操作和分解来进行。在本节, 我们介绍矩阵的基本概念、矩阵的基本运算、特征值与特征向量, 以及矩阵的一些常用分解, 如基于特征值与特征向量的分解、奇异值分解和主成分分析。

3.3.1 基本概念

1. 矩阵的定义

$m \times n$ 个数 $a_{ij} (i=1, 2, \dots, m; j=1, 2, \dots, n)$ 构成的如下排列称为 m 行 n 列的矩阵, 简称 $m \times n$ 矩阵:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \quad (3-78)$$

矩阵一般用大写字母表示。例如, 上述矩阵可以简记为 $A = (a_{ij})_{m \times n}$ 。特别地, 当 $m = n$ 时, 矩阵 $(a_{ij})_{m \times n}$ 称为 m 阶方阵。

对于矩阵 $A = (a_{ij})_{m \times n}$, 把矩阵 $B = (a_{ji})_{n \times m}$ 称为 A 的转置矩阵, 记为 $B = A^T$:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix} \quad (3-79)$$

2. 特殊矩阵

方阵 $A = (a_{ij})_{n \times n}$ 有如下几种常见的特殊类型。

- 对称矩阵: 若 $\forall i, j \in \{1, 2, \dots, n\}, a_{ij} = a_{ji}$, 则称 A 为对称矩阵;
- 对角矩阵: 若 $\forall i \neq j \in \{1, 2, \dots, n\}, a_{ij} = 0$, 则称 A 为对角矩阵;

$$A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix} \quad (3-80)$$

- 单位矩阵: 若 $\forall i \neq j \in \{1, 2, \cdots, n\}, a_{ij} = 0$ 且 $\forall k \in \{1, 2, \cdots, n\}, a_{kk} = 1$, 则称 A 为单位矩阵, 通常记为 $A=I$ 或 $A=E$:

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{bmatrix} \quad (3-81)$$

- 正交矩阵: 若 $A^T A = A A^T = I$, 则称 A 为正交矩阵; 将矩阵 A 记为 $A = [a_1, a_2, \cdots, a_n]$, 这里 a_i 是 A 的第 i 列, 则有

$$A^T A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{bmatrix} [a_1, a_2, \cdots, a_n] = \begin{bmatrix} a_1^T a_1 & a_1^T a_2 & \cdots & a_1^T a_n \\ a_2^T a_1 & a_2^T a_2 & \cdots & a_2^T a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n^T a_1 & \cdots & a_n^T a_{n-1} & a_n^T a_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{bmatrix} \quad (3-82)$$

$$\Leftrightarrow a_i^T a_j = \begin{cases} 1, & \text{如果 } i = j \\ 0, & \text{如果 } i \neq j \end{cases}$$

类似地, 我们利用 $A A^T = I$ 也可证明 A 的每行也具有类似的性质。

- 正定矩阵: 若对任意列向量 $v \neq 0$, 都有 $v^T A v > 0$, 则称 A 为正定矩阵。
- 半正定矩阵: 若对任意列向量 $v \neq 0$, 都有 $v^T A v \geq 0$, 则称 A 为半正定矩阵。
- 可逆矩阵: 存在方阵 B , 使得 $AB = BA = I$, 则称 A 为可逆矩阵, 称 $B = A^{-1}$ 为 A 的逆矩阵。如果矩阵 A 不可逆, 则称 A 为奇异矩阵。

3. 矩阵的行列式

在介绍行列式之前, 首先引入排列的奇偶性概念: 假设在 $1, 2, \cdots, n$ 的某种排列 p_1, p_2, \cdots, p_n 中, $\forall k \in [1, n]$, 有 i_k 个比 p_k 小的数排在 p_k 之后, 则用 $\sum_{k=1}^n i_k$ 的奇偶性来描述排列 p_1, p_2, \cdots, p_n 的奇偶性。例如, 对于排列 $1, 3, 2, 7, 5$, $\sum_{k=1}^5 i_k = 0 + 1 + 0 + 1 + 0 = 2$, 则该排列为偶排列。

设 $1, 2, \cdots, n$ 的所有排列集合为 Ω , 称 $\det(A) = |A| = \sum_{p_1, p_2, \cdots, p_n \in \Omega} (\sum_{j=1}^n (-1)^{\sum_{k=1}^n i_k} \cdot a_{jp_j})$ 为方阵 $A = (a_{ij})_{n \times n}$ 的行列式, 这里 $\sum_{k=1}^n i_k$ 的含义同上。

4. 秩

若将矩阵 $A = (a_{ij})_{m \times n}$ 的每一行、每一列都看做一个向量, 则线性无关的行向量最大个数称为 A 的行秩, 线性无关的列向量最大个数称为 A 的列秩。矩阵 A 的行秩等于列秩。

5. 向量的范数

对于一个 d 维向量 $\mathbf{v} = [v_1, v_2, \dots, v_d] \in \mathbb{R}^d$, 其 $p(p \geq 1)$ 范数定义为

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^d |v_i|^p \right)^{\frac{1}{p}} \quad (3-83)$$

根据该定义, 可以得到如下常用的 1-范数 (也称为 L_1 范数):

$$\|\mathbf{v}\|_1 = \sum_{i=1}^d |v_i| \quad (3-84)$$

相应地, 2-范数 (也称为 L_2 范数) 定义为

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^d v_i^2} \quad (3-85)$$

范数的概念也可以引申到矩阵中。感兴趣的读者可以阅读矩阵计算的相关教材[21]。在本书中, 我们只介绍向量的范数。

3.3.2 基本运算

本节介绍矩阵的和/差、数乘、乘法、幂等基本运算。

1. 和/差运算

对任意矩阵 $\mathbf{A} = (a_{ij})_{m \times n}$ 和 $\mathbf{B} = (b_{ij})_{m \times n}$, 有

$$\mathbf{A} \pm \mathbf{B} = \begin{bmatrix} a_{11} \pm b_{11} & \cdots & a_{1n} \pm b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} \pm b_{m1} & \cdots & a_{mn} \pm b_{mn} \end{bmatrix} \quad (3-86)$$

不难证明, 矩阵的和运算满足交换律和结合律。

2. 数乘运算

对任意矩阵 $\mathbf{A} = (a_{ij})_{m \times n}$ 和常数 k , 有

$$k\mathbf{A} = \begin{bmatrix} ka_{11} & \cdots & ka_{1n} \\ \vdots & \ddots & \vdots \\ ka_{m1} & \cdots & ka_{mn} \end{bmatrix} \quad (3-87)$$

不难证明, 矩阵的数乘运算满足分配律。

3. 乘法运算

对任意矩阵 $\mathbf{A} = (a_{ij})_{m \times n}$ 和 $\mathbf{B} = (b_{ij})_{n \times p}$, \mathbf{A} 与 \mathbf{B} 的乘积 \mathbf{C} 是一个 $m \times p$ 矩阵, 且 $\forall i, j \in \{1, 2, \dots, m\}$, 有 $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$:

$$\begin{aligned}
 C = AB &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix} \\
 &= \begin{bmatrix} \sum_{k=1}^n a_{1k}b_{k1} & \sum_{k=1}^n a_{1k}b_{k2} & \cdots & \sum_{k=1}^n a_{1k}b_{kp} \\ \sum_{k=1}^n a_{2k}b_{k1} & \sum_{k=1}^n a_{2k}b_{k2} & \cdots & \sum_{k=1}^n a_{2k}b_{kp} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^n a_{mk}b_{k1} & \sum_{k=1}^n a_{mk}b_{k2} & \cdots & \sum_{k=1}^n a_{mk}b_{kp} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{bmatrix} \quad (3-88)
 \end{aligned}$$

不难证明, 乘法运算满足结合律和分配律。

4. 幂运算

对任意方阵 $A = (a_{ij})_{n \times n}$ 和正整数 k , A 的 k 次幂定义为

$$A^k = \underbrace{A \cdots A}_{k \text{ 个 } A} \quad (3-89)$$

对任意正整数 k, m , 矩阵 A 的幂运算满足:

$$\begin{cases} A^k A^m = A^{k+m} \\ (A^k)^m = A^{km} \end{cases} \quad (3-90)$$

3.3.3 特征值与特征向量

对于方阵 $A = (a_{ij})_{n \times n}$, 若存在 n 维列向量 $v \neq 0$, 使得:

$$Av = \lambda v \quad (3-91)$$

则称 v 为 A 的特征向量, 称 λ 为对应的特征值。本节介绍求解特征值与特征向量的方程方法与幂方法。

1. 方程法求解

求解特征值与特征向量时, 可以把式 (3-91) 等价写为

$$(\lambda I - A)v = 0 \quad (3-92)$$

可以直接求解该方程; 或者通过求解 $|\lambda I - A| = 0$ 来求解特征值 λ , 进而代入式 (3-92) 求得对应的特征向量 v 。下面通过一个例子加以说明。

例 3-2 求矩阵

$$A = \begin{bmatrix} 6 & 4 \\ 1 & 3 \end{bmatrix} \quad (3-93)$$

的特征值与特征向量。

解 设所求特征值与特征向量分别为 λ 和 \mathbf{v} , 则有

$$|\lambda I - A| = \begin{vmatrix} \lambda - 6 & -4 \\ -1 & \lambda - 3 \end{vmatrix} = (\lambda - 6)(\lambda - 3) - 4 = 0 \quad (3-94)$$

解得 $\lambda = 2$ 或 $\lambda = 7$ 。

当 $\lambda = 2$ 时, 方程 $(\lambda I - A)\mathbf{v} = 0$ 变为

$$\begin{bmatrix} -4 & -4 \\ -1 & -1 \end{bmatrix} \mathbf{v} = 0 \quad (3-95)$$

解得 $\mathbf{v} = k_1 \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, $k_1 \neq 0$ 。

当 $\lambda = 7$ 时, 方程 $(\lambda I - A)\mathbf{v} = 0$ 变为

$$\begin{bmatrix} 1 & -4 \\ -1 & 4 \end{bmatrix} \mathbf{v} = 0 \quad (3-96)$$

解得 $\mathbf{v} = k_2 \begin{pmatrix} 4 \\ 1 \end{pmatrix}$, $k_2 \neq 0$ 。

由方程法可以看出, 在复数域上, 方阵 A 的特征值个数一定等于方阵的阶数。

2. 幂方法求解

对于方阵 $A = (a_{ij})_{n \times n}$, 假设其 n 个特征值按模由大到小的顺序依次排列为 $\lambda_1, \lambda_2, \dots, \lambda_n$, 对应的特征向量分别为 $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ 。任取特征向量的线性组合 $\mathbf{w}_0 = \sum_{i=1}^n c_i \mathbf{v}_i$ ($c_i \neq 0$), 然后构造序列 $\mathbf{w}_1 = A\mathbf{w}_0, \mathbf{w}_2 = A\mathbf{w}_1 = A^2\mathbf{w}_0, \dots, \mathbf{w}_k = A\mathbf{w}_{k-1} = A^k\mathbf{w}_0, \dots$, 不难看出:

$$\mathbf{w}_k = A^k \sum_{i=1}^n c_i \mathbf{v}_i = A^{k-1} \sum_{i=1}^n c_i (\lambda_i \mathbf{v}_i) = \dots = \sum_{i=1}^n c_i \lambda_i^k \mathbf{v}_i \quad (3-97)$$

式 (3-97) 可改写为

$$\mathbf{w}_k = c_1 \lambda_1^k \mathbf{v}_1 + \lambda_1^k \sum_{i=2}^n c_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \mathbf{v}_i \quad (3-98)$$

若 $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$, 即 $|\lambda_1|$ 严格大于 $|\lambda_2|$, 则有

$$\lim_{k \rightarrow +\infty} \left(\frac{\lambda_i}{\lambda_1} \right)^k = 0, i \geq 2 \Rightarrow \lim_{k \rightarrow +\infty} \mathbf{w}_k = c_1 \lambda_1^k \mathbf{v}_1 \quad (3-99)$$

从而 $\lambda_1 = \lim_{k \rightarrow +\infty} \frac{w_k}{w_{k-1}}$ 。由于 λ_1 是方阵 A 模最大的特征值, 因此称其为方阵 A 的主特征值 (principal eigenvalue), 把对应的特征向量 v_1 称为方阵 A 的主特征向量 (principal eigenvector)。由式 (3-99) 可得

$$v_1 = c \lim_{k \rightarrow +\infty} \frac{w_k}{\|w_k\|_2} \quad (3-100)$$

这里 c 为常数。因此, 主特征值与主特征向量均可以通过迭代的方法近似求得。

从这里的分析我们可以看出, 幂方法收敛速度依赖于 $|\lambda_2|/|\lambda_1|$ 。 $|\lambda_2|/|\lambda_1|$ 越小, 幂方法收敛的速度越快。当前两个特征值 λ_1 和 λ_2 满足 $|\lambda_1| = |\lambda_2|$ 时, 我们可以分 3 种情况讨论:

- $\lambda_1 = \lambda_2 \in \mathbb{R}$, 上面讨论的算法仍然收敛;
- $\lambda_1 = -\lambda_2 \in \mathbb{R}$, 只需要对矩阵 A^2 使用幂方法即可;
- 当 λ_1, λ_2 是共轭复数时, 幂方法不收敛。

在这里我们只是简单讨论幂方法的基本原理。当 λ_1, λ_2 是共轭复数时, 我们可以使用幂方法更加复杂的变体来求解。

在具体迭代时, 对于方阵 $A = (a_{ij})_{n \times n}$, 任取初始特征向量 $w_0 \neq 0$ 并指定一个小正数 ε , 然后 $\forall k \geq 1$ 可通过下面的式 (3-101) 迭代得到最小的 k 值:

$$\begin{aligned} w_k &= \frac{Aw_{k-1}}{\|Aw_{k-1}\|_2} \\ \text{s.t. } \|Aw_k - Aw_{k-1}\|_2 &< \varepsilon \end{aligned} \quad (3-101)$$

这样得到的向量 w_k 称为方阵 A 的 (近似) 主特征向量, 相应的特征值称为 (近似) 主特征值。

这里仍以例 3-2 为例, 若我们选取初始特征向量 $w_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ 并指定小正数 $\varepsilon = 0.05$, 则有

$$\begin{aligned} w_1 &= \frac{Aw_0}{\|Aw_0\|_2} = \frac{1}{\sqrt{29}} \begin{bmatrix} 5 \\ 2 \end{bmatrix} \approx \begin{bmatrix} 0.9285 \\ 0.3714 \end{bmatrix} \\ w_2 &= \frac{Aw_1}{\|Aw_1\|_2} = \frac{1}{\sqrt{1565}} \begin{bmatrix} 38 \\ 11 \end{bmatrix} \approx \begin{bmatrix} 0.9606 \\ 0.2781 \end{bmatrix} \\ w_3 &= \frac{Aw_2}{\|Aw_2\|_2} = \frac{1}{5\sqrt{3161}} \begin{bmatrix} 272 \\ 71 \end{bmatrix} \approx \begin{bmatrix} 0.9676 \\ 0.2526 \end{bmatrix} \\ w_4 &= \frac{Aw_3}{\|Aw_3\|_2} = \frac{1}{\sqrt{3906281}} \begin{bmatrix} 1916 \\ 485 \end{bmatrix} \approx \begin{bmatrix} 0.9694 \\ 0.2454 \end{bmatrix} \\ &\vdots \end{aligned}$$

计算到 w_4 时我们发现, $\|Aw_4 - Aw_3\|_2 \approx 0.0264 < 0.05$, 满足条件, 于是 w_4 为方阵 A 的 (近似) 主特征向量。根据式 (3-101), 特征向量 w_k 恒为单位向量, 因此 $\lambda = w_k^T \lambda w_k$, 于是根据式 (3-91) 有

$$\lambda = \mathbf{w}_k^T \lambda \mathbf{w}_k = \mathbf{w}_k^T \mathbf{A} \mathbf{w}_k \quad (3-102)$$

因此, 方阵 \mathbf{A} 对应于 \mathbf{w}_4 的 (近似) 主特征值为

$$\lambda_4 = \mathbf{w}_4^T \mathbf{A} \mathbf{w}_4 = \frac{1}{3906281^2} [1916 \ 485] \begin{bmatrix} 6 & 4 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 1916 \\ 485 \end{bmatrix} \approx 7.0088$$

这个结果跟例 3-2 的其中一个结果非常接近。如果要计算另一组特征值和特征向量, 可构造方阵 $\mathbf{B} = \mathbf{A} - \lambda_4 \mathbf{w}_4 \mathbf{w}_4^T$, 然后对 \mathbf{B} 进行迭代计算, 过程从略。

3.3.4 矩阵分解

1. 基于特征值与特征向量的分解

设方阵 $\mathbf{A} = (a_{ij})_{n \times n}$ 的 n 个特征值分别为 $\lambda_1, \lambda_2, \dots, \lambda_n$, 对应的特征向量分别为 $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ 。根据特征值和特征向量的定义有

$$\mathbf{A} \mathbf{v}_i = \lambda_i \mathbf{v}_i, i = 1, 2, \dots, n \quad (3-103)$$

我们可以将这 n 个等式写成矩阵的形式:

$$\mathbf{A}[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n] = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n] \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \quad (3-104)$$

若将 $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n]$ 记为 \mathbf{V} , 将 $n \times n$ 的对角矩阵 $\begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$ 记为 \mathbf{D} :

$$\mathbf{D} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n) = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \quad (3-105)$$

则式 (3-104) 可写为

$$\mathbf{A} \mathbf{V} = \mathbf{V} \mathbf{D} \quad (3-106)$$

如果矩阵 \mathbf{V} 可逆, 可以将矩阵 \mathbf{A} 表示为

$$\mathbf{A} = \mathbf{V} \mathbf{D} \mathbf{V}^{-1} \quad (3-107)$$

若矩阵 \mathbf{A} 是对称矩阵, 则矩阵 \mathbf{V} 是正交矩阵 (意味着 $\mathbf{V}^T \mathbf{V} = \mathbf{V} \mathbf{V}^T = \mathbf{I}$, 即 $\mathbf{V}^{-1} = \mathbf{V}^T$), 可以将矩阵 \mathbf{A} 表示为

$$A = VDV^T \quad (3-108)$$

2. 奇异值分解

在前面的特征值分解中，对于一个实矩阵，所得的特征值和特征向量中可能还有虚数。下面我们介绍奇异值分解 (singular value decomposition, SVD)。奇异值分解在矩阵计算中是一项非常有用的工具，在机器学习特别是降维中有着广泛的应用。在 SVD 中，我们也把一个实矩阵分解成为 3 个实矩阵的乘积，且中间的那个矩阵是对角矩阵。

假设矩阵 $A = (a_{ij})_{m \times n}$ 的秩为 r ，则存在矩阵 $U = (u_{ij})_{m \times r}$ 、 $\Sigma = (\sigma_{ij})_{r \times r}$ 、 $V = (v_{ij})_{n \times r}$ ，使得

$$A = U\Sigma V^T \quad (3-109)$$

这里 $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r)$ ($\sigma_i \geq 0$) 为对角矩阵， $U = [u_1, u_2, \dots, u_r]$ 和 $V = [v_1, v_2, \dots, v_r]$ 满足：

$$u_i^T u_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \quad v_i^T v_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad (3-110)$$

我们把式 (3-109) 称为矩阵 A 的奇异值分解，把 u_i 称为左奇异向量，把 v_i 称为右奇异向量，把 σ_i 称为奇异值。图 3-4 显示了 SVD 的示例。注意，根据秩的定义，我们知道 $r \leq \min(m, n)$ 。

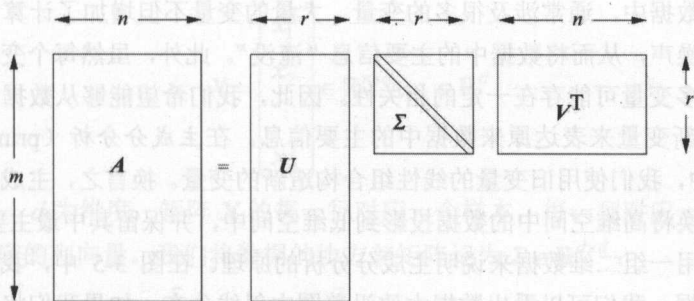


图 3-4 SVD 示例

式 (3-109) 可展开为

$$A = U\Sigma V^T = [u_1, u_2, \dots, u_r] \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_r^T \end{bmatrix} = \sum_{i=1}^r \sigma_i u_i v_i^T \quad (3-111)$$

尽管对角矩阵可以任意排列 $\sigma_1, \sigma_2, \dots, \sigma_r$ 的次序 (调整 U 、 V 中对应列的顺序即可)，但在矩阵分解中，我们一般约定 $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$ 。

下面我们给出一个 SVD 的实际例子:

$$\begin{bmatrix} 1 & 3 & 0 \\ 0 & -4 & 10 \\ 3 & 8 & 12 \\ 4 & 0 & 13 \end{bmatrix} = \begin{bmatrix} 0.03 & -0.34 & 0.20 \\ 0.43 & 0.65 & -0.53 \\ 0.64 & -0.64 & -0.39 \\ 0.64 & 0.23 & 0.73 \end{bmatrix} \begin{bmatrix} 21.03 & 0 & 0 \\ 0 & 9.02 & 0 \\ 0 & 0 & 2.04 \end{bmatrix} \begin{bmatrix} 0.21 & 0.17 & 0.96 \\ -0.15 & -0.97 & 0.20 \\ 0.97 & -0.19 & -0.18 \end{bmatrix}$$

可以通过对向量的变换来解释奇异值分解。假设要对向量 w 进行线性变换 Aw :

$$Aw = U\Sigma V^T w = \sum_{i=1}^r \sigma_i u_i v_i^T w = \sum_{i=1}^r (\sigma_i v_i^T w) u_i \quad (3-112)$$

我们可以分别考虑每个 $\sigma_i u_i v_i^T w$ 。首先利用 v_i 对 w 进行变换, 得到 w 在 v_i 方向上的分量 $v_i^T w$ (这是一个实数值); 然后使用奇异值 σ_i 来调整该分量的大小, 得到 $\sigma_i v_i^T w$; 最后将该分量投影到由 u_i 规定的方向上, 得到 $(\sigma_i v_i^T w) u_i$ 。将所有分量累加起来, 就得到 $\sum_{i=1}^r \sigma_i u_i v_i^T w = Aw$ 。

简单地讲, V 的列 v_i 可以是“输入”空间的基向量, U 的列 u_i 可以认为是输出空间的基向量, 而对应的奇异值 σ_i 可以认为是对输入空间和输出空间的向量长度进行控制的项。

3.3.5 主成分分析

在很多实际数据中, 通常涉及很多的变量。大量的变量不但增加了计算复杂度, 而且有些变量有可能是噪声, 从而将数据中的主要信息“淹没”。此外, 虽然每个变量都提供了相应的信息, 但是很多变量可能存在一定的相关性。因此, 我们希望能够从数据中提取最主要的信息, 用较少的新变量来表达原来数据中的主要信息。在主成分分析 (principal component analysis, PCA) 中, 我们使用旧变量的线性组合构造新的变量。换言之, 主成分分析的基本思想是利用线性变换将高维空间中的数据投影到低维空间中, 并保留其中最主要的信息。

下面我们使用一组二维数据来说明主成分分析的原理。在图 3-5 中, 我们有一组二维数据。对于这组数据, 我们可以看出数据大致沿着图中斜线分布。如果我们将数据投影到这条斜线上, 则数据的分散程度最大, 能够给我们提供最多的关于数据间差异的信息。严格地讲, 投影后数据的分散程度最大, 即方差最大。注意, 我们这里是将数据投影到一条直线上, 如果用 v_1 、 v_2 分别表示原始数据中的两个变量, 则可以使用这两个变量的线性组合来表示所构造的新变量 v :

$$v = av_1 + bv_2 + c \quad (3-113)$$

这里 $a, b, c \in \mathbb{R}$ 。我们将使得所得投影后新变量 v 方差最大的 v 称为第一主成分。

在上面的这个二维数据的例子中, 我们将数据投影到了一个二维空间, 因此仅需要第一主成分。对于更加高维的数据, 如果我们要将其投影到多维空间中, 那么需要多个主成分。在得到第一主成分之后, 我们求解第二主成分时要求其第一主成分正交, 表示要从原始数

据中提取第一主成分没有提取到的信息。一般而言，在求解第 i 个主成分时，要求它与前面已知的 $i-1$ 个主成分都正交。

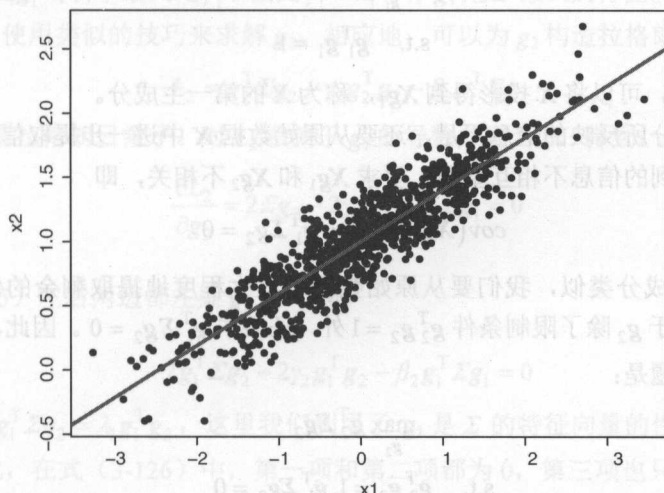


图 3-5 PCA 示例（图中斜线表示一组二维数据的第一主成分的方向）

在下面的讨论中，我们使用严格的数学语言来讨论对于给定的数据，如何求解主成分。

假设待研究的数据可表示为 $n \times d$ 的矩阵 X ：

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}, \mathbf{x}_i \in \mathbb{R}^d \quad (3-114)$$

这里 n 是样本数， d 为维度，矩阵 X 的每一行对应一个样本，每一列对应一个变量，而 \mathbf{x}_i 是第 i 个样本所对应的列向量。我们将数据的协方差矩阵记为 $\Sigma \in \mathbb{R}^{d \times d}$ ：

$$\Sigma = \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \cdots & \sigma_{dd} \end{bmatrix} \in \mathbb{R}^{d \times d}, \text{ 且 } \Sigma = \Sigma^T \quad (3-115)$$

在主成分分析中，我们使用线性变换将数据投影到低维空间中，希望能够保存原来数据的主要信息。在前面的讨论中，我们知道在主成分分析中希望投影后数据点的分散度更高，即投影后的方差最大。使用线性变换 \mathbf{g}_1 将 X 投影到一维空间后，新的数据表示为 $X\mathbf{g}_1$ ，对应的方差为

$$\text{var}(X\mathbf{g}_1) = \mathbf{g}_1^T \Sigma \mathbf{g}_1 \quad (3-116)$$

因此，在主成分分析中，我们需要最大化 $\mathbf{g}_1^T \Sigma \mathbf{g}_1$ 。注意，我们必须对 \mathbf{g}_1 加以限制，否则 $\mathbf{g}_1^T \Sigma \mathbf{g}_1$ 无界。这里我们假设 \mathbf{g}_1 具有单位长度，即要求 $\mathbf{g}_1^T \mathbf{g}_1 = 1$ 。因此，我们需要求解如下的

优化问题:

$$\begin{aligned} \max_{\mathbf{g}_1} \quad & \mathbf{g}_1^T \Sigma \mathbf{g}_1 \\ \text{s.t.} \quad & \mathbf{g}_1^T \mathbf{g}_1 = 1 \end{aligned} \quad (3-117)$$

得到投影向量 \mathbf{g}_1 后, 可以将 \mathbf{X} 投影得到 $\mathbf{X}\mathbf{g}_1$, 称为 \mathbf{X} 的第一主成分。

如果第一主成分所反映的信息不足, 还要从原始数据 \mathbf{X} 中进一步提取信息。计算第二主成分时, 为了使得到的信息不相互重叠, 要求 $\mathbf{X}\mathbf{g}_1$ 和 $\mathbf{X}\mathbf{g}_2$ 不相关, 即

$$\text{cov}(\mathbf{X}\mathbf{g}_1, \mathbf{X}\mathbf{g}_2) = \mathbf{g}_1^T \Sigma \mathbf{g}_2 = 0 \quad (3-118)$$

与求解第一主成分类似, 我们要从原始数据中最大程度地提取剩余的信息, 要最大化 $\mathbf{g}_2^T \Sigma \mathbf{g}_2$ 。但是, 对于 \mathbf{g}_2 除了限制条件 $\mathbf{g}_2^T \mathbf{g}_2 = 1$ 外, 还要求 $\mathbf{g}_1^T \Sigma \mathbf{g}_2 = 0$ 。因此, 对于 \mathbf{g}_2 , 我们需要求解的优化问题是:

$$\begin{aligned} \max_{\mathbf{g}_2} \quad & \mathbf{g}_2^T \Sigma \mathbf{g}_2 \\ \text{s.t.} \quad & \mathbf{g}_2^T \mathbf{g}_2 = 1, \mathbf{g}_1^T \Sigma \mathbf{g}_2 = 0 \end{aligned} \quad (3-119)$$

一般来说, 在求解第 i 个主成分时, 我们要最大化 $\mathbf{g}_i^T \Sigma \mathbf{g}_i$, 同时要求 $\mathbf{X}\mathbf{g}_i$ 和 $\mathbf{X}\mathbf{g}_j (j = 1, 2, \dots, i-1)$ 都不相关。因此, 相应的优化问题是:

$$\begin{aligned} \max_{\mathbf{g}_i} \quad & \mathbf{g}_i^T \Sigma \mathbf{g}_i \\ \text{s.t.} \quad & \mathbf{g}_i^T \mathbf{g}_i = 1, \mathbf{g}_i^T \Sigma \mathbf{g}_j = 0, j = 1, 2, \dots, i-1 \end{aligned} \quad (3-120)$$

下面介绍两种方法来求解 \mathbf{g}_i 。注意, 协方差矩阵 Σ 半正定且对称, 因此其所有的特征值都是非负实数。这里我们把 Σ 的 d 个特征值记为 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d \geq 0$, 并将其对应的特征向量记为 $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d$ 。这些特征向量也可以写成矩阵的形式 $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d] \in \mathbb{R}^{d \times d}$ 。下面我们严格证明 \mathbf{g}_i 是协方差矩阵 Σ 的第 i 大的特征值所对应的特征向量 \mathbf{v}_i 。

1. 拉格朗日函数法

第一种方法是使用拉格朗日函数来求解条件约束的最优化问题。求解 \mathbf{g}_1 时, 我们可以构造相应的拉格朗日函数 L_1 :

$$L_1 = \mathbf{g}_1^T \Sigma \mathbf{g}_1 - \gamma_1 \mathbf{g}_1^T \mathbf{g}_1 \quad (3-121)$$

这里 $\gamma_1 \in \mathbb{R}$ 是拉格朗日乘子。对 L_1 求关于 \mathbf{g}_1 的导数并置为 0^①, 得到:

$$\frac{\partial L_1}{\partial \mathbf{g}_1} = 2\Sigma \mathbf{g}_1 - 2\gamma_1 \mathbf{g}_1 = 0 \Rightarrow \Sigma \mathbf{g}_1 = \gamma_1 \mathbf{g}_1 \quad (3-122)$$

这个结果表明 \mathbf{g}_1 是 Σ 的一个特征向量, 且其对应的特征值是 γ_1 。重新回到目标函数, 我们有:

① 关于如何对向量或者矩阵求导, 推荐参考文献[22]。

$$\mathbf{g}_1^T \Sigma \mathbf{g}_1 = \mathbf{g}_1^T (\gamma_1 \mathbf{g}_1) = \gamma_1 (\mathbf{g}_1^T \mathbf{g}_1) = \gamma_1 \quad (3-123)$$

所以, 最大化 $\mathbf{g}_1^T \Sigma \mathbf{g}_1$ 等价于最大化 γ_1 。因此 $\gamma_1 = \lambda_1$, 即 \mathbf{g}_1 是 Σ 最大的特征值所对应的特征向量。

接下来我们使用类似的技巧来求解 \mathbf{g}_2 。相应地, 可以为 \mathbf{g}_2 构造拉格朗日函数 L_2 :

$$L_2 = \mathbf{g}_2^T \Sigma \mathbf{g}_2 - \gamma_2 \mathbf{g}_2^T \mathbf{g}_2 - \beta_2 \mathbf{g}_1^T \Sigma \mathbf{g}_2 \quad (3-124)$$

这里 $\gamma_2, \beta_2 \in \mathbb{R}$ 是拉格朗日乘子。对 L_2 求关于 \mathbf{g}_2 的导数并置为 0, 得到:

$$\frac{\partial L_2}{\partial \mathbf{g}_2} = 2\Sigma \mathbf{g}_2 - 2\gamma_2 \mathbf{g}_2 - \beta_2 \Sigma \mathbf{g}_1 = 0 \quad (3-125)$$

将式 (3-125) 左右两边都左乘 \mathbf{g}_1^T , 可得:

$$2\mathbf{g}_1^T \Sigma \mathbf{g}_2 - 2\gamma_2 \mathbf{g}_1^T \mathbf{g}_2 - \beta_2 \mathbf{g}_1^T \Sigma \mathbf{g}_1 = 0 \quad (3-126)$$

注意到 $0 = \mathbf{g}_1^T \Sigma \mathbf{g}_2 = \lambda_1 \mathbf{g}_1^T \mathbf{g}_2$, 这里我们利用了 \mathbf{g}_1 是 Σ 的特征向量的性质, 所以可以确定 $\mathbf{g}_1^T \mathbf{g}_2 = 0$ 。因此, 在式 (3-126) 中, 第一项和第二项都为 0, 第三项也只能为 0:

$$0 = \beta_2 \mathbf{g}_1^T \Sigma \mathbf{g}_1 = \beta_2 \lambda_1 \mathbf{g}_1^T \mathbf{g}_1 \Rightarrow \beta_2 = 0 \quad (3-127)$$

将 $\beta_2 = 0$ 代回到 $\frac{\partial L_2}{\partial \mathbf{g}_2}$ 中, 我们得到:

$$\frac{\partial L_2}{\partial \mathbf{g}_2} = 2\Sigma \mathbf{g}_2 - 2\gamma_2 \mathbf{g}_2 = 0 \Rightarrow \Sigma \mathbf{g}_2 = \gamma_2 \mathbf{g}_2 \quad (3-128)$$

所以 \mathbf{g}_2 也是 Σ 的特征向量, 其对应的特征值为 γ_2 。使用类似的技巧, 我们可以得到 $\gamma_2 = \lambda_2$ 。因此, \mathbf{g}_2 是 Σ 第二大的特征值所对应的特征向量。

我们可以顺次使用上面的推导, 获得一般性的结论: \mathbf{g}_i 是 Σ 第 i ($i=1, 2, \dots, d$) 大的特征值所对应的特征向量。

2. 特征值分解法

第二种方法是直接利用协方差矩阵 Σ 的特征值分解。我们知道

$$\Sigma \mathbf{v}_i = \lambda_i \mathbf{v}_i \quad (3-129)$$

写成矩阵的形式就是:

$$\begin{aligned} \Sigma [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d] &= [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d] \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d) \\ \Rightarrow \Sigma \mathbf{V} &= \mathbf{V} \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d) \end{aligned} \quad (3-130)$$

这里 $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d]$ 。由于矩阵 Σ 是对称矩阵, 因此:

$$\mathbf{V}^T \mathbf{V} = \mathbf{I}_d \quad (3-131)$$

这里 \mathbf{I}_d 是 $d \times d$ 的单位矩阵。利用该性质和式 (3-130), 我们有

$$\mathbf{V}^T \Sigma \mathbf{V} = \mathbf{V}^T \mathbf{V} \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d) = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d) \quad (3-132)$$

注意到 V 构成了 d 维空间的一组基, 因此可以将 g_1 写成:

$$g_1 = Vw_1 \quad (3-133)$$

这里 w_1 是 d 维向量 $w_1 = [w_{11}, w_{12}, \dots, w_{1d}]^T$ 。在求解 g_1 时, 我们要最大化 $g_1^T \Sigma g_1$, 它可以进一步表示为:

$$\begin{aligned} g_1^T \Sigma g_1 &= w_1^T V^T \Sigma V w_1 = w_1^T (V^T \Sigma V) w_1 \\ &= w_1^T \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d) w_1 = \sum_{i=1}^d \lambda_i w_{1i}^2 \end{aligned} \quad (3-134)$$

注意到关于向量 g_1 有约束条件 $g_1^T g_1 = 1$, 可以表示为:

$$w_1^T V^T V w_1 = 1 \Rightarrow w_1^T w_1 = 1 \Leftrightarrow \sum_{i=1}^d w_{1i}^2 = 1 \quad (3-135)$$

注意到 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d \geq 0$ 。我们要在满足 $\sum_{i=1}^d w_{1i}^2 = 1$ 的条件下最大化 $\sum_{i=1}^d \lambda_i w_{1i}^2$, 对于最优解必须有:

$$w_{11} = 1, w_{12} = \dots = w_{1d} = 0 \quad (3-136)$$

此时有:

$$g_1 = [v_1, v_2, \dots, v_d][1, 0, \dots, 0]^T = v_1 \quad (3-137)$$

因此, g_1 就是 Σ 最大的特征值所对应的特征向量。

接下来考虑 g_2 的求解。类似地, 可以将 g_2 表示为 $g_2 = Vw_2$, 这里 w_2 是 d 维向量 $w_2 = [w_{21}, w_{22}, \dots, w_{2d}]^T$ 。在求解 g_2 时, 我们要最大化 $g_2^T \Sigma g_2$, 它也可表示为:

$$\begin{aligned} g_2^T \Sigma g_2 &= w_2^T V^T \Sigma V w_2 = w_2^T (V^T \Sigma V) w_2 \\ &= w_2^T \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d) w_2 = \sum_{i=1}^d \lambda_i w_{2i}^2 \end{aligned} \quad (3-138)$$

但是要注意到关于 g_2 还有两个约束条件。与关于 g_1 的推导类似, 其中一个约束条件可以表示为:

$$g_2^T g_2 = 1 \Rightarrow \sum_{i=1}^d w_{2i}^2 = 1 \quad (3-139)$$

这里讨论另一个约束条件 $g_1^T \Sigma g_2 = 0$:

$$\begin{aligned} g_1^T \Sigma g_2 &= 0 \Leftrightarrow (\Sigma g_1)^T g_2 = 0 \Leftrightarrow \lambda_1 g_1^T g_2 = 0 \Rightarrow \lambda_1 v_1^T g_2 = 0 \\ &\Rightarrow \lambda_1 v_1^T V w_2 = 0 \Rightarrow \lambda_1 v_1^T [v_1, v_2, \dots, v_d] w_2 = 0 \\ &\Rightarrow \lambda_1 [1, 0, \dots, 0] [w_{21}, w_{22}, \dots, w_{2d}]^T = 0 \\ &\Rightarrow \lambda_1 w_{21} = 0 \Rightarrow w_{21} = 0 \end{aligned} \quad (3-140)$$

因此, 约束条件 $\sum_{i=1}^d w_{2i}^2 = 1$ 可以表示成:

$$\sum_{i=2}^d w_{2i}^2 = 1 \quad (3-141)$$

而我们要最大化的目标 $\mathbf{g}_2^T \Sigma \mathbf{g}_2$ 也可以表示为:

$$\mathbf{g}_2^T \Sigma \mathbf{g}_2 = \sum_{i=1}^d \lambda_i w_{2i}^2 = \sum_{i=2}^d \lambda_i w_{2i}^2 \quad (3-142)$$

在这种情况下, 要取得最大值, 只有:

$$w_{22} = 1, w_{23} = \cdots = w_{2d} = 0 \quad (3-143)$$

此时有:

$$\mathbf{g}_2 = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_d] [0, 1, 0, \cdots, 0]^T = \mathbf{v}_2 \quad (3-144)$$

因此, \mathbf{g}_2 就是 Σ 第二大的特征值所对应的特征向量。

类似地, 我们可以顺次使用上面的推导, 获得一般性的结论: \mathbf{g}_i 是 Σ 第 i ($i=1, 2, \cdots, d$) 大的特征值所对应的特征向量。

3. 贡献率

根据前面的推导, 第 i 个主成分所对应的方差为:

$$\text{var}(\mathbf{X}\mathbf{g}_i) = \mathbf{g}_i^T \Sigma \mathbf{g}_i = \lambda_i \quad (3-145)$$

假设我们从原始数据中选取了所有 d 个主成分, 则所得的主成分的总方差为:

$$\sum_{i=1}^d \text{var}(\mathbf{X}\mathbf{g}_i) = \sum_{i=1}^d \lambda_i \quad (3-146)$$

我们将 $\frac{\lambda_i}{\sum_{j=1}^d \lambda_j}$ 称为第 i 个主成分的贡献率, 描述了第 i 个主成分提取的信息占有所有信息的

比例。将 $\frac{\sum_{j=1}^i \lambda_j}{\sum_{j=1}^d \lambda_j}$ 称为前 i 个主成分的累积贡献率, 描述了前 i 个主成分提取的所有信息的比例。

在实际中, 对于高维数据, 通常要选择提取的主成分的数目。通常当选取的主成分的累积贡献率达到较高的比例 (如 80%) 时即可。

例 3-3 假设有一组三维数据对应的协方差矩阵如下, 要求出对应的所有 3 个主成分:

$$\Sigma = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 2.24 \\ 0 & 2.24 & 2 \end{bmatrix} \quad (3-147)$$

解 我们将原始数据中的 3 个变量记为 d_1 、 d_2 、 d_3 。可以求得 Σ 的特征值和对应的特征

向量如下:

$$\lambda_1 = 6.45, v_1 = [0, 1, 0]$$

$$\lambda_2 = 2.00, v_2 = [0.839, 0, -0.544]$$

$$\lambda_3 = 1.55, v_3 = [0.544, 0, 0.839]$$

则第一个主成分对应的新变量是:

$$0 \times d_1 + 1 \times d_2 + 0 \times d_3 = d_2$$

类似地, 第二个主成分对应的新变量是:

$$0.839 \times d_1 + 0 \times d_2 - 0.544 \times d_3 = 0.839d_1 - 0.544d_3$$

如果我们只选取前面两个主成分, 则其对应的累积贡献率为:

$$\frac{6.45 + 2.00}{6.45 + 2.00 + 1.55} = 84.53\%$$

3.3.6 R 中矩阵的计算

在本节, 我们将要介绍如何使用 R 中提供的函数来进行较复杂的矩阵计算和分解, 如 SVD 和 PCA。关于在 R 中进行矩阵计算的基本知识可参见 2.4.3 节。在本节中, 着重介绍 eigen 函数和 svd 函数, 同时也介绍在矩阵操作中常用的 diag 函数。本节所有的矩阵分析相关的程序都在文件 matrix_decomp_example.R 中, 而如何使用 PCA 的程序在文件 PCA_example.R 中。

我们介绍 matrix_decomp_example.R 文件以演示如何在 R 中进行矩阵分解时, 首先使用 rnorm 函数产生一系列基于正态分布的随机数。具体来说, rnorm(N) 产生 N 个随机数。为了保证读者多次运行程序能够得到一致的结果, 我们首先使用 set.seed 函数将随机种子设为 1 (读者可以根据需要设置不同的随机种子)。这样, 我们得到了一个 4×4 的矩阵 A 和一个 4×3 的矩阵 B。

```
set.seed(1)
A = matrix(rnorm(16), 4, 4)
B = matrix(rnorm(12), 4, 3)
```

然后演示 diag 函数的用法。R 中 diag 函数的主要使用方法如下。

- diag(A): 如果输入参数是一个矩阵 A, 则返回其主对角线上的元素组成的一个向量。
- diag(v): 如果输入参数是一个向量, 则返回一个方阵, 其主对角线上的元素来自向量 v, 其他元素为 0。
- diag(k): 返回一个 k×k 的单位矩阵。
- diag(a, k): 返回一个 k×k 的对角矩阵, 且对角线的元素都是 a。

注意, 在 R 中我们还可以将 diag(A) 作为左值, 从而修改矩阵 A 的对角线上的元素。我

们使用如下代码将 A 的对角线元素都乘以 2，将 B 的对角线元素都加 5。

```
diag(A) <- diag(A) * 2
diag(B) <- diag(B) + 5
```

接着使用 eigen 函数计算矩阵 A 的特征值分解。该函数的主要用法如下：

- eigen(A)：计算矩阵 A 的特征值和特征向量。
- eigen(A, only.values=T)：只计算矩阵 A 的特征值。
- eigen(A, symmetric=T)：指定矩阵 A 为对称矩阵并计算其特征值和特征向量。

该函数返回的计算结果是一个列表，其中 values 保存特征值，vectors 保留对应的特征向量。

具体来说，vectors 部分是一个矩阵，每列对应一个特征向量。

先计算矩阵 A 的特征值和特征向量，其代码如下：

```
E_A=eigen(A)
E_A$values
E_A$vectors
# Verify the decomposition
Delta_A = E_A$vectors %*% diag(E_A$values) %*% solve(E_A$vectors) - A
```

计算的特征值如下：

```
> E_A$values
[1] 3.038531+0.000000i -0.639326+1.598516i -0.639326-1.598516i
[4] -1.720029+0.000000i
```

计算的特征向量如下：

```
> E_A$vectors
      [,1]      [,2]      [,3]
[1,] 0.1005331+0i -0.3117119+0.1413634i -0.3117119-0.1413634i
[2,] -0.1273888+0i -0.7008545+0.0000000i -0.7008545+0.0000000i
[3,] 0.9763473+0i -0.0082256-0.1360593i -0.0082256+0.1360593i
[4,] 0.1428676+0i 0.2922527+0.5363428i 0.2922527-0.5363428i
      [,4]
[1,] 0.39879838+0i
[2,] -0.90121557+0i
[3,] 0.16814566+0i
[4,] -0.02230215+0i
```

为了验证对应的特征值分解，上面计算了矩阵 Delta_A，这里 solve(E_A\$vectors) 计算特征向量组成的矩阵的逆矩阵。矩阵 Delta_A 为：

```
> Delta_A
      [,1]      [,2]
[1,] -1.110223e-15+5.25347e-17i -4.996004e-16+2.94439e-17i
[2,] -1.054712e-15+2.28771e-16i -6.661338e-16+5.16959e-17i
[3,] 1.110223e-16-5.67661e-17i -8.881784e-16-6.22705e-18i
[4,] 1.332268e-15-1.10856e-16i 7.771561e-16-6.94834e-17i
      [,3]      [,4]
```

```
[1,] 4.440892e-16+2.14516e-17i 2.220446e-16+5.639643e-17i
[2,] -3.885781e-16+2.36165e-17i -4.440892e-16+1.090217e-16i
[3,] 8.881784e-16+1.15951e-17i -2.220446e-16-5.513789e-17i
[4,] -6.106227e-16-5.61287e-17i 4.440892e-16-1.110718e-16i
```

可以看出 Delta_A 非常接近零矩阵。

使用 `eigen(A, only.values = T)` 时只计算特征值。

```
E_A_only = eigen(A, only.values = T)
```

在这种情况下, `E_A_only$vectors` 为 NULL。

```
> E_A_only$vectors
NULL
```

接下来生成一个对称矩阵, 并计算它的特征值和特征向量, 同时验证对应的特征值分解。

```
AA = A + t(A)
E_AA = eigen(AA, symmetric=T)
E_AA$values
E_AA$vectors
Delta_AA = E_AA$vectors %*% diag(E_AA$values) %*% t(E_AA$vectors) - AA
```

对应的输出如下:

```
> E_AA$values
[1] 6.398773 0.311260 -2.265899 -4.364432
> E_AA$vectors
      [,1]      [,2]      [,3]      [,4]
[1,] -0.004510515 -0.2627321 0.85543559 0.44629748
[2,] -0.016587737 0.3347921 0.51450720 -0.78925372
[3,] 0.973683278 0.2099321 0.02524274 0.08504233
[4,] 0.227256191 -0.8802355 -0.05361995 -0.41311613
> Delta_AA
      [,1]      [,2]      [,3]      [,4]
[1,] -8.881784e-16 -2.220446e-16 1.110223e-16 -3.330669e-16
[2,] -4.440892e-16 -5.329071e-15 -2.886580e-15 5.107026e-15
[3,] 1.110223e-16 -2.886580e-15 -8.881784e-16 3.774758e-15
[4,] -3.330669e-16 5.107026e-15 3.774758e-15 -7.799317e-15
```

可以看出, 对称矩阵 AA 的特征值和特征向量都是实数。由于 `E_AA$vectors` 是正交矩阵, 因此, 在验证特征值分解时, 直接使用了 `t(E_AA$vectors)` 而不是 `E_AA$vectors` 的逆矩阵。

然后, 我们可以直接使用 `svd` 函数来计算矩阵的奇异值分解。返回的结果也是一个列表对象, 其中 `u` 对应 SVD 中的 *U* 矩阵、`v` 对应 SVD 中的 *V* 矩阵, 而 `d` 是一个向量, 保存着所有的奇异值。

```
S_B=svd(B)
S_B$u
S_B$v
S_B$d
Delta_B = S_B$u %*% diag(S_B$d) %*% t(S_B$v) - B
```

对应的输出如下:

```
> S_B$u
      [,1]      [,2]      [,3]
[1,] -0.5091763  0.36701422  0.7482381
[2,] -0.7646891 -0.52790019 -0.1667094
[3,] -0.2938966  0.76529501 -0.4904828
[4,]  0.2638465  0.03074893  0.4144568
> S_B$v
      [,1]      [,2]      [,3]
[1,] -0.4980551  0.3829775  0.7779906
[2,] -0.8096673 -0.5265935 -0.2591105
[3,] -0.3104513  0.7589648 -0.5723569
> S_B$d
[1] 6.714185 5.163792 4.389618
> Delta_B
      [,1]      [,2]      [,3]
[1,] 2.664535e-15 -4.773959e-15 -2.220446e-15
[2,] -9.992007e-16  8.881784e-16 -1.151856e-15
[3,]  0.000000e+00 -8.049117e-16  5.329071e-15
[4,]  9.992007e-16 -6.661338e-16 -1.332268e-15
```

接下来我们讨论如何在 R 中计算 PCA。在文件 PCA_example.R 的例子中,使用了 iris 数据。在第 6 章中我们会更加详细地介绍该数据集,这里只是用该数据集中的前 4 个变量进行 PCA 分解。这里我们使用 R 中的 prcomp 函数来进行主成分分析。代码如下:

```
data(iris)
D = iris
# show the basic information of the data
str(D)

# log transform
D_log <- log(D[, 1:4])

# apply PCA
M_pca <- prcomp(D_log, center = TRUE, scale. = TRUE)

# print method
print(M_pca)
# members of M_pca
M_pca$sdev
M_pca$rotation

# plot method
plot(M_pca, type = "l")

# summary method
summary(M_pca)
```



```
# Predict PCs
y_test <- predict(M_pca, newdata= D_log[1:10,])
```

由于 R 中已经提供了 iris 数据, 因此这里我们首先直接使用 data 函数载入该数据, 并使用 str 函数对该数据集进行简单分析。这里我们只考虑该数据的前面 4 列, 同时将其进行了对数变换, 同时将数据保存在数据框 D_log 中。

之后我们使用 prcomp 函数对输入数据 D_log 进行主成分分析。这里我们将 center 和 scale. 都设为真, 表示要将数据进行标准化处理, 使得每个变量处理后均值为 0、方差为 1。如果不进行标准化处理, 那么可能有些变量由于值比较大在计算中占优势, 从而使得 PCA 产生偏差。

得到 PCA 模型 M_pca 后, 可以使用 print 函数打印该模型的主要信息, 其输出如下:

```
> print(M_pca)
Standard deviations:
[1] 1.7124583 0.9523797 0.3647029 0.1656840

Rotation:
          PC1          PC2          PC3          PC4
Sepal.Length 0.5038236 -0.45499872 0.7088547 0.19147575
Sepal.Width -0.3023682 -0.88914419 -0.3311628 -0.09125405
Petal.Length 0.5767881 -0.03378802 -0.2192793 -0.78618732
Petal.Width 0.5674952 -0.03545628 -0.5829003 0.58044745
```

这里产生了所有的 4 个主成分。print 函数打印出主成分的标准差和协方差的特征向量。进一步, 利用 M_pca\$sdev, 可以得到每个主成分的标准差, 即协方差矩阵的特征值的平方根。利用 M_pca\$rotation, 可以得到协方差矩阵对应的特征向量。直接使用 M_pca\$sdev 和 M_pca\$rotation 的输出如下:

```
> M_pca$sdev
[1] 1.7124583 0.9523797 0.3647029 0.1656840
> M_pca$rotation
          PC1          PC2          PC3          PC4
Sepal.Length 0.5038236 -0.45499872 0.7088547 0.19147575
Sepal.Width -0.3023682 -0.88914419 -0.3311628 -0.09125405
Petal.Length 0.5767881 -0.03378802 -0.2192793 -0.78618732
Petal.Width 0.5674952 -0.03545628 -0.5829003 0.58044745
```

可以看出, 其输出与 print 函数类似, 但是我们单独输出了每个部分。利用 M_pca\$rotation, 可以从原始数据得到新的主成分。例如, 对于第一主成分 PC1, 可以利用下面的公式得到:

```
PC1=0.5038236×Sepal.Length-0.3023682×Sepal.Width+0.5767881×Petal.Length+
0.5674952×Petal.Width
```

其中的系数来自于 M_pca\$rotation 的第一列, 这里 Sepal.Length、Sepal.Width、Petal.Length 和 Petal.Width 是 4 个原始变量。

利用 plot 函数, 可以得到所有主成分对应的方差。图 3-6 是模型 M_pca 对应的输出, 其中 x 轴是主成分的编号, y 轴是对应的方差。也可以使用 M_pca\$sdev^2 来直接计算各个主成

分所对应的方差。使用图 3-6, 可以简单地估计一下应该保留多少个主成分来做进一步的分析。从图 3-6 可以看出, 对于 iris 数据, 两个主成分就可以了。

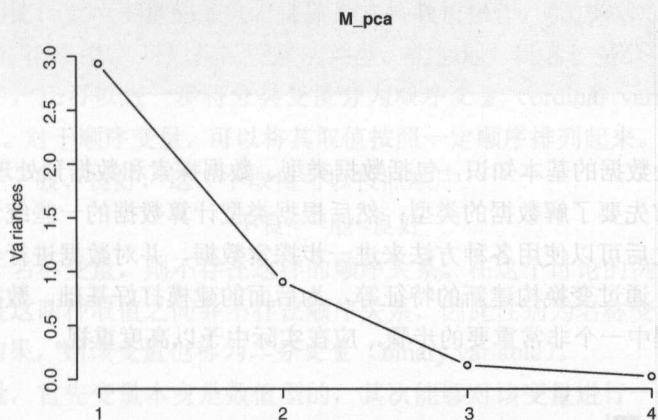


图 3-6 PCA 中各个主成分所对应的方差

利用 summary 函数可以进一步研究每个主成分的重要性。其输出的第一行是每个主成分对应的标准差（平方后为方差），第二行为每个主成分的贡献率，第三行为累积贡献率。从 summary 函数的输出可以看出，前两个主成分已经贡献了 95.99% 的方差，因此，选用前两个主成分对于 iris 数据就足够了。

```
> summary(M_pca)
Importance of components:

          PC1      PC2      PC3      PC4
Standard deviation  1.7125 0.9524 0.36470 0.16568
Proportion of Variance 0.7331 0.2268 0.03325 0.00686
Cumulative Proportion 0.7331 0.9599 0.99314 1.00000
```

最后，使用 predict 函数来对新数据进行主成分分析。在使用过程中，利用 newdata 来指定新的数据。在我们的代码中，直接使用了 D_log 的前 10 行。predict 函数的输出是这 10 个样本所对应的 4 个主成分，其输出如下：

```
> y_test
```

	PC1	PC2	PC3	PC4
1	-2.406639	-0.3969554	0.19396467	0.004779476
2	-2.223539	0.6901804	0.35000151	0.048868378
3	-2.581105	0.4275418	0.01889761	0.049909545
4	-2.450869	0.6860074	-0.06874595	-0.149646465
5	-2.536853	-0.5082516	0.02932259	-0.040048202
6	-1.841495	-1.2899381	-0.25276831	0.163890597
7	-2.479490	0.1011323	-0.49740434	0.122759047
8	-2.348593	-0.1569003	0.13601863	-0.095498156
9	-2.535948	1.2477681	-0.11188119	-0.075468614
10	-2.625580	0.5074073	0.65947371	-0.473259070

第4章 数据探索和预处理

本章主要讨论数据的基本知识,包括数据类型、数据探索和数据预处理。在实际问题中,得到数据之后,首先要了解数据的类型;然后根据类型计算数据的一些统计量以了解每个特征的大致信息;之后可以使用各种方法来进一步探索数据,并对数据进行一些预处理,如删除一些无关变量,通过变换构建新的特征等,为后面的建模打好基础。数据探索和预处理是解决实际问题过程中一个非常重要的步骤,应在实际中予以高度重视。

4.1 数据类型

在实际问题中,数据呈现出各种各样的形式。在本书中,我们将数据看成是一组对象(object)的集合,而每个对象由一定数目的特征(feature)或者属性(attribute)来描述。根据不同的语境,对象也可称为样本(sample)、记录(record)、数据点(data point)、向量(vector)等;特征也可称为变量(variable)。每个对象可能有不同数目的特征,但在机器学习中,为了处理的方便,我们一般都假设对象的特征是一致的。

本书用 n 表示对象或者样本的数目, d 表示特征的数目,同时我们把第 i 个样本记为

$$\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{id}]^T \quad (4-1)$$

这里 \mathbf{x}_i 是 d 维列向量。在本书中,我们通常将向量写成列向量的形式。

下面我们讨论特征。由于实际问题的复杂性,特征有不同的类型。这里举一个搜集学生体检信息的例子。简单起见,我们记录了表 4-1 所示的数据。

表 4-1 学生体检数据及范例

学号	姓名	性别	年龄	体重	身高
201503345	李乐	男	21	65	175

在表 4-1 中,有 6 个特征,包括学号、姓名、性别、年龄、体重和身高。下面逐一分析各个特征,并介绍特征的不同类型。

首先分析学号。在这个例子中,学号虽然是整数,但是我们并不能执行加法或者减法运算。对于体重,65 (kg) 加 1 有着明确的含义,而某个学号加 1 并不一定有着明确的含义(甚至有可能是空号)。因此,在这个例子中,学号虽然是整数,但是我们并不能将其视为数值变量。

在数据分析中主要有两类变量:分类变量(categorical variable)和数值变量(numeric variable)。分类变量的取值来自一个集合,顾名思义,每一个取值表示该变量的一个状态(或者分类)。在这个讨论学生体检的例子中,性别就是一个最明显的例子,其取值集合是{男,

女}。在该数据集中,学号也是一个分类变量,其取值集合就是该校所有学号的集合。而数值变量的取值一般是整数域或者实数域,当然,根据数据的实际情况,其取值区间会有具体变化。与分类变量相比,数值变量的最大特征就是支持数值操作,如加减法。因此,通过考虑能否对数据进行加减法操作,可以推断变量的类型。类似地,姓名也是分类变量。

在分类变量中,还可以进一步将分类变量分为顺序变量 (ordinal variable) 和名称变量 (nominal variable)。对于顺序变量,可以将其取值按照一定顺序排列起来。例如,我们评价体检的结果是不良、一般、良好,这 3 个取值可以按照顺序

不良<一般<良好

排列起来。而对于名称变量,则不存在这样的顺序关系。在这个讨论的例子中,学生的性别为男或者女,而且这两种取值之间并不存在顺序关系,因此性别为名称变量。如果分类变量的取值只有两种结果,则该变量也称为二分变量 (binary variable)。

对于数值变量,首先变量本身是数值型的,其次能够对该变量进行一些常用的适用于数值变量的操作,如计算平均值和标准差等。在判定一个变量是数值变量还是分类变量时,一个简单的方法就是考虑能否在该变量上进行数值操作。在上面的例子中,学生年龄是数值变量,可以计算学生的平均年龄和年龄的标准差。而对于学号变量来说,虽然在这个例子中该变量都是整数型的,我们也可以计算学号的平均值,但是这个平均值没有任何意义。因此,我们断定学号变量是分类变量,而不是数值变量。

类似地,我们可以看出身高和体重变量都是数值变量。

在很多实际应用中,都有一组对象,而每个对象都可以用一组属性或者变量来表示。如果每个对象的属性一致,数据一般可以表示成矩阵的形式,称为数据矩阵 (data matrix)。在数据矩阵中,每行是一个数据点 (或者一个对象),每列是一个变量、属性或特征。例如,在前面讨论的体检数据中,每一行对应一个学生的体检数据,每一列是体检中的一个变量。当然,我们也可以将每一列表示成一个对象,而每一行表示成一个属性。从矩阵操作上讲,只是将矩阵做了一个简单的转置。因此,在以后的讨论中,我们约定每行是一个对象,每列是一个属性。

在实际中,很多这类数据都是用文本文件来保存的。利用关系数据库也能很好地表示数据矩阵。常见的文本文件格式包括 csv (comma separated values) 和 tsv (tab separated values)。在 R 中,我们可以使用 read.csv 或者类似的 read.table 函数来直接将数据导入到内存中。我们在下一节中将会详细地介绍如何在 R 中导入数据和进行基本的数据探索。

在实际应用中,还有各种各样的其他数据。例如,社交网络就以图 (graph) 的形式保存。在本书中,我们主要讨论以数据矩阵保存的数据。

4.2 数据探索

在实际中,数据探索 (data exploration) 是了解数据的一个很重要的步骤。我们经常说要根据数据的具体特征选择不同的方法,而数据探索就是了解数据具体特征的重要步骤。在数

据探索中，我们主要计算数据的一些统计量，并通过图和表的形式进行总结。在 4.4 节，我们将详细讨论多种以图的形式表现数据主要特征的方法。本节主要介绍在数据探索中常用的统计量和 R 中的计算方法。

4.2.1 常用统计量

一般来讲，得到数据后要首先检查数据质量 (data quality)。例如，每个变量的取值是否都是合乎数据定义的。通常，我们可以通过计算每个变量的一些统计量来检查数据是否存在问题。另一个问题是数据中通常存在缺失值 (missing value)。缺失值可能由很多原因引起，如在数据收集过程中，有些数据可能因难以获得而被标识为缺失值。因此，在进行数据探索时要计算每个变量是否存在缺失值，以及缺失值的比例等。

计算统计量是探索数据的一种非常直接且有效的方法。统计量包括两方面：

- (1) 单个变量的统计量，如数值变量的均值、极值，分类变量的所有不同取值等；
- (2) 变量之间的统计量，如每两个变量之间的相关系数。

此外，对于不同类型的变量，我们需要计算不同的统计量。在一般的数据分析中，对于分类变量，我们通常感兴趣的有：

- 有多少个不同的取值；
- 每个取值的频率；
- 最常见的取值。

对于数值变量，我们可以计算各种不同的统计量，包括

- 均值；
- 方差和标准差；
- 中位数；
- 下四分位数；
- 上四分位数；
- 最小值；
- 最大值；
- 偏度；
- 数据的具体分布等。

这些统计量在数学基础的相关章节中已经作了详细讨论。接下来，我们介绍如何在 R 中利用相关的函数来探索数据。

4.2.2 使用 R 实际探索数据

下面我们利用具体的例子讨论如何在 R 中导入文本数据，并通过计算相关的统计量来探索数据。本节讨论的 R 程序都在文件 `R_data_load_example.R` 中。

我们前面讨论的数据一般都可以保存在文本文件中，其中应用最广泛的是 csv 文件。在

csv 文件中，文件的每一行对应于数据矩阵的每一行，每列之间以逗号（,）分隔。csv 文件也有一些变体，如 tsv 文件，其主要区别在于使用制表符 Tab 键（\t）来分隔各列。

R 中提供了 `read.csv` 函数和 `read.table` 函数，可以直接将数据从 csv 文件导入为数据框。下面是一段示例 R 程序。这段程序从 `Data/iris.data` 文件导入数据，并进行了简单的探索。

```
data_file = 'Data/iris.data'
D = read.csv(data_file, header=F)
# Get key statistics for this data frame
# show some samples of the data frame D
print('the first 10 rows of D is')
print(head(D, 10))
print('the last 10 rows of D')
print(tail(D, 10))
# Show the summary
print('summary of D')
print(summary(D))
# Check the type of the 1st column
c1 = class(D[,1])
msg = paste0('the type of 1st column is ', c1)
print(msg)
c5 = class(D[,5])
msg = paste0('the type of the 5th column is ', c5)
print(msg)

# An alternative function to load data
D1 = read.table(data_file, header=F, sep=',')
print('summary of D1')
print(summary(D1))
```

首先我们使用 `read.csv` 导入文件。在使用该函数时，只需要指定文件名即可。注意，这里将 `header` 设为 `F`，表示该 csv 文件的第一行就是数据，没有各列的列名信息。将数据读入并保存在数据框 `D` 中之后，使用 `head` 函数来输出前 10 行，使用 `tail` 函数来输出最后 10 行。在探索数据时，需要直接检查原始数据。虽然在很多时候由于数据规模太大而无法检查所有数据，但是直接检查一部分原始数据永远是必要的。

我们直接使用了 R 中的 `summary` 函数来计算数据框中每列的统计量。对于数值变量，`summary` 函数计算如下统计量：

- 最小值；
- 下四分位数；
- 中位数；
- 均值；
- 上四分位数；
- 最大值。

对于分类变量，`summary` 函数值列出了所有不同的取值及每个取值出现的频率。

在上面这段程序中,还检查了这个数据框第1列和第5列的类型。在R中,可以使用class函数来得到变量的类型。从输出可以看出,第1列是数值变量,而第5列是factor类型,表示是分类变量。

在R中,read.csv专门用来导入csv文件。read.table是更加通用的函数,使用它可以轻松地处理tsv和csv文件。在上面的程序中,我们也示范了如何使用read.table函数来导入csv函数。在使用时,我们通过sep=', '来指定分隔符是逗号。

下面是这段R程序的输出:

```
[1] "the first 10 rows of D is"
      V1 V2 V3 V4      V5
1  5.1 3.5 1.4 0.2 Iris-setosa
2  4.9 3.0 1.4 0.2 Iris-setosa
3  4.7 3.2 1.3 0.2 Iris-setosa
4  4.6 3.1 1.5 0.2 Iris-setosa
5  5.0 3.6 1.4 0.2 Iris-setosa
6  5.4 3.9 1.7 0.4 Iris-setosa
7  4.6 3.4 1.4 0.3 Iris-setosa
8  5.0 3.4 1.5 0.2 Iris-setosa
9  4.4 2.9 1.4 0.2 Iris-setosa
10 4.9 3.1 1.5 0.1 Iris-setosa
[1] "the last 10 rows of D"
      V1 V2 V3 V4      V5
141 6.7 3.1 5.6 2.4 Iris-virginica
142 6.9 3.1 5.1 2.3 Iris-virginica
143 5.8 2.7 5.1 1.9 Iris-virginica
144 6.8 3.2 5.9 2.3 Iris-virginica
145 6.7 3.3 5.7 2.5 Iris-virginica
146 6.7 3.0 5.2 2.3 Iris-virginica
147 6.3 2.5 5.0 1.9 Iris-virginica
148 6.5 3.0 5.2 2.0 Iris-virginica
149 6.2 3.4 5.4 2.3 Iris-virginica
150 5.9 3.0 5.1 1.8 Iris-virginica
[1] "summary of D"
      V1      V2      V3      V4      V5
Min.   :4.300 Min.   :2.000 Min.   :1.000 Min.   :0.100 Iris-setosa :50
1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300 Iris-versicolor:50
Median :5.800 Median :3.000 Median :4.350 Median :1.300 Iris-virginica :50
Mean   :5.843 Mean   :3.054 Mean   :3.759 Mean   :1.199
3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
Max.   :7.900 Max.   :4.400 Max.   :6.900 Max.   :2.500
[1] "the type of 1st column is numeric"
[1] "the type of the 5th column is factor"
[1] "summary of D1"
      V1      V2      V3      V4      V5
Min.   :4.300 Min.   :2.000 Min.   :1.000 Min.   :0.100 Iris-setosa :50
1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300 Iris-versicolor:50
```

```

Median :5.800 Median :3.000 Median :4.350 Median :1.300 Iris-virginica :50
Mean :5.843 Mean :3.054 Mean :3.759 Mean :1.199
3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500

```

我们也可以直接从互联网上读取 csv 数据。R 的一大优点是有非常多的第三方包，提供了丰富的功能和算法。在这里，我们可以使用 Rcurl 包从网络直接读取数据。在下面的示例程序中，首先检查 Rcurl 包有没有安装，如果没有安装，则使用 install.packages 函数安装该包。安装完该包之后，使用 library 函数载入该包。f_URL 指定了我们所要读取的文件的 URL。我们首先使用 Rcurl 包中的 getURL 函数来读取该文件，并保存在一个字符串 f_text 中；然后使用 read.csv 函数将字符串转换为数据框。注意，这次在使用 read.csv 函数时，我们必须显式指定输入数据是 text 类型。

```

# Step 2. Load data from a URL directly
# We need to install Rcurl package first.
# Before we install it, we need to check if the Rcurl package is already installed.
RCurl.installed <- 'RCurl' %in% rownames(installed.packages())
if (RCurl.installed) {
  print("the Rcurl package is already installed, let's load it...")
} else {
  print("let's install the Rcurl package first...")
  install.packages('RCurl', dependencies=T)
}
library('RCurl')
# Use Rcurl to read the file specified by the URL f_URL
f_URL = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
# f_text is the string we download for this URL
f_text = getURL(f_URL)
D2 = read.csv(text=f_text, header=F)
print('summary of D2')
print(summary(D2))

```

对应的输出如下：

```

[1] "the Rcurl package is already installed, let's load it..."
[1] "summary of D2"

```

	V1	V2	V3	V4	V5
Min.	:4.300	Min. :2.000	Min. :1.000	Min. :0.100	Iris-setosa :50
1st Qu.:	:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	Iris-versicolor:50
Median :	:5.800	Median :3.000	Median :4.350	Median :1.300	Iris-virginica :50
Mean :	:5.843	Mean :3.054	Mean :3.759	Mean :1.199	
3rd Qu.:	:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :	:7.900	Max. :4.400	Max. :6.900	Max. :2.500	

R 中也自带了一些数据。对于这些自带的数据库，可以使用 data 函数直接载入。使用 data()

可以列出 R 中所有的自带数据。指定数据集名,则可载入相应的数据。在下面的程序中,首先使用 `data()` 列出所有的自带数据,然后使用 `data('mtcars')` 载入 `mtcars` 数据,最后使用 `summary` 函数计算这个数据集的主要统计量。

```
# Step 3. Load dataset provided by R
# list all data sets available in R
data()
# load the mtcars dataset
data('mtcars')
print('summary of mtcars')
print(summary(mtcars))
```

下面是关于 `mtcars` 数据集的输出:

```
[1] "summary of mtcars"
      mpg          cyl          disp         hp        drat         wt
Min.   :10.40  Min.   :4.000  Min.   : 71.1  Min.   : 52.0  Min.   :2.760  Min.   :1.513
1st Qu.:15.43  1st Qu.:4.000  1st Qu.:120.8  1st Qu.: 96.5  1st Qu.:3.080  1st Qu.:2.581
Median :19.20  Median :6.000  Median :196.3  Median :123.0  Median :3.695  Median :3.325
Mean   :20.09  Mean   :6.188  Mean   :230.7  Mean   :146.7  Mean   :3.597  Mean   :3.217
3rd Qu.:22.80  3rd Qu.:8.000  3rd Qu.:326.0  3rd Qu.:180.0  3rd Qu.:3.920  3rd Qu.:3.610
Max.   :33.90  Max.   :8.000  Max.   :472.0  Max.   :335.0  Max.   :4.930  Max.   :5.424

      qsec          vs          am         gear        carb
Min.   :14.50  Min.   :0.0000  Min.   :0.0000  Min.   :3.000  Min.   :1.000
1st Qu.:16.89  1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:3.000  1st Qu.:2.000
Median :17.71  Median :0.0000  Median :0.0000  Median :4.000  Median :2.000
Mean   :17.85  Mean   :0.4375  Mean   :0.4062  Mean   :3.688  Mean   :2.812
3rd Qu.:18.90  3rd Qu.:1.0000  3rd Qu.:1.0000  3rd Qu.:4.000  3rd Qu.:4.000
Max.   :22.90  Max.   :1.0000  Max.   :1.0000  Max.   :5.000  Max.   :8.000
```

虽然 `summary` 函数可以一次计算多个统计量,但是在一些场合我们只需要计算某一个特定统计量。表 4-2 列出了常用的统计函数及它们对应的功能。

表 4-2 R 中常用的统计函数

函数名	描述
<code>min</code>	计算最小值
<code>max</code>	计算最大值
<code>mean</code>	计算均值
<code>median</code>	计算中位数
<code>quantile</code>	计算四分位数
<code>skewness</code>	计算偏度
<code>table</code>	计算每种取值(单变量时)或每种取值组合(多变量时)出现的频率

在表 4-2 中,除 `skewness` 函数外,其他函数在基本的 R 中直接提供了,因此不需要安

装第三方包。为了使用 `skewness` 函数计算偏度，我们需要载入 `moments` 包。

下面的程序将会示范如何使用这些函数计算第 1 列数据对应的统计量。此外，我们还会利用第 5 列数据示范如何使用 `table` 函数。与前面安装 `RCurl` 包类似，这里将安装 `moments` 包以使用 `skewness` 函数。

```
# Step 4. Instead of using the summary function, we compute these statistics
# using separate functions
m1 = min(D[,1])
msg = paste0('min = ', m1)
print(msg)
m3 = max(D[,1])
msg = paste0('max = ', m3)
print(msg)
m2 = mean(D[,1])
msg = paste0('mean = ', m2)
print(msg)
md = median(D[,1])
msg = paste0('median = ', md)
print(msg)
q1 = quantile(D[,1], 0.25)
msg = paste0('25% quantile = ', q1)
print(msg)
q2 = quantile(D[,1], 0.5)
msg = paste0('50% quantile = ', q2)
print(msg)
q3 = quantile(D[,1], 0.75)
msg = paste0('75% quantile = ', q3)
print(msg)
# Check the skewness of a single column in R
# We need to install moments package first.
# Before we install it, we need to check if the moments package is already installed.
moments.installed <- 'moments' %in% rownames(installed.packages())
if (moments.installed) {
  print("the moments package is already installed, let's load it...")
}else {
  print("let's install the moments package first...")
  install.packages('moments', dependencies=T)
}
library('moments')
s1 = skewness((D[,1]))
msg = paste0('the skewness of 1st column is ', s1)
print(s1)
```

```
print('the distribution of the 5th column')
t5 = table(D[,5])
print(t5)
```

对应的输出如下：

```
[1] "min = 4.3"
[1] "max = 7.9"
[1] "mean = 5.84333333333333"
[1] "median = 5.8"
[1] "25% quantile = 5.1"
[1] "50% quantile = 5.8"
[1] "75% quantile = 6.4"
[1] "the moments package is already installed, let's load it..."
[1] 0.3117531
[1] "the distribution of the 5th column"
```

```
Iris-setosa Iris-versicolor Iris-virginica
          50           50           50
```

4.3 数据预处理

在实际建模中，数据预处理（data preprocessing）是非常关键的一步，直接影响最终模型结果的好坏。在大多数情况下，原始数据都不宜直接用来建模，需要对数据进行多方面的预处理之后才能进入建模环节。一般而言，数据的预处理包括：

- （1）删除部分数据，如可以直接删除冗余或者无关数据；
- （2）增加新的数据，从已有数据中，可以构造新的特征；
- （3）数据的变换，原有的数据不适合直接建模，需要作一些变换以便模型可以更好地直接利用。

当然，数据的预处理和具体使用哪种模型建模直接相关。好的数据预处理能够显著提高模型的性能，坏的数据预处理能够完全毁掉模型的性能。此外，不同的模型对于数据的要求可能完全不一样。基于树的一些模型对于数据不是特别敏感，但另外一些模型，如线性回归，对于数据则较为敏感。

由于数据的预处理与实际原始数据的特征和所要解决的问题直接相关，因此，在本节中，我们着重讨论一些一般性的技巧。

4.3.1 缺失值的处理

在实际中，缺失值是很普遍的。有些缺失值可能是因为搜集数据时有遗漏或者无法搜集，有些缺失数据可能是暂时的（当时无法搜集但是过段时间后可能得到）。在我们后面讨论的算法中，有些算法能够直接处理缺失值，如基于决策树的算法；而有些算法不能直接处理缺

失值,如线性回归。

在实际中,对于缺失数据处理的第一步是:明确缺失数据的重要性。如果缺失变量对于目标值的预测不重要,那么可以直接删除该变量;如果缺失变量很重要,无法直接丢弃,那么通常用如下两种方法来处理。第一种方法是采用能够直接处理缺失数据的模型来进行建模,如基于决策树的模型等。如果我们要使用那些不能直接处理缺失数据的算法进行建模,同时也不能直接丢弃缺失变量,那么可以采用第二种方法——进行缺失值填充。

每一个缺失值的填充可以视为一个单独的预测问题:我们可以利用其他没有缺失的变量来估计缺失的变量。这相当于为了解决原始的预测问题,我们引入了一个新的预测问题。这种方法显著地增加了问题的复杂度。事实上,我们最终关心的并不是缺失值的估计是否准确,而是原始问题的预测是否准确。因此,在解决缺失值填充问题时,我们更加倾向于使用简单的方法来估计缺失值。这样做一方面能够降低缺失值估计的计算复杂度,另一方面还能够避免过拟合。

缺失值填充有如下常用方法。

(1) 使用平均值或者中位数来进行填充。

(2) 使用 k 近邻方法进行填充。具体来说,假设样本 \mathbf{x}_i 的第 j 个变量缺失(记为 x_{ij}),我们的目标就是要估计 x_{ij} 。首先利用 \mathbf{x}_i 中没有缺失的变量,找到最相似的 k 个样本,并用这 k 个样本的第 j 个变量的平均值作为 x_{ij} 的估计值。在该方法中,主要的控制参数是 k ,即邻域的大小。在实践中人们发现,缺失值填充算法的性能常常对 k 不敏感。

4.3.2 数据的标准化

对于数值变量,每个变量都有自己的单位。在处理数据时我们需要同时考虑多个变量。例如,我们要计算两个向量 $\mathbf{x}_1 = [x_{11}, x_{12}]^T$ 、 $\mathbf{x}_2 = [x_{21}, x_{22}]^T$ 的欧几里得距离:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2} \quad (4-2)$$

如果第一个变量的绝对差比第二个变量的绝对差大很多,那么欧几里得距离 $d(\mathbf{x}_1, \mathbf{x}_2)$ 就会被第一个变量所左右,而第二个变量被忽视。

为了解决这个问题,通常可以先进行数据标准化。对于每个变量 j ,我们可以计算其对应的均值 μ_j 和标准差 σ_j , 然后进行如下变换:

$$x_{ij}^* = \frac{x_{ij} - \mu_j}{\sigma_j} \quad (4-3)$$

经过数据标准化后,所有的变量对应的均值都是 0,标准差都是 1。经过标准化处理之后的 x_{ij}^* 也称为 Z 分值 (Z-score), 因此,该变换也称为 Z 分值标准化 (Z-score normalization)。

注意,均值和标准差的计算都受到离群数据的影响。如果某个变量对应的离群数据较多,那么可以将上面变换公式中的均值 μ_j 和标准差 σ_j 替换为受离群数据影响较小的统计量。例

如, 均值 μ_j 可以直接替换为中位数 (median), 标准差 σ_j 可以替换为绝对平均偏差 (absolute average deviation, AAD):

$$AAD_j = \frac{1}{n} \sum_{i=1}^n |x_{ij} - \mu_j| \quad (4-4)$$

这里我们先不讨论离群数据, 本章稍后将予以详细介绍。

在 3.2.1 节中我们已经介绍过绝对平均值。第 3 章介绍的其他关于数据分布的集中位置和分散程度的统计量都可以在数据标准化中使用。

在 R 中, 我们可以利用 scale 函数来完成对矩阵类数据的标准化。假设我们利用 scale 函数来对 R 中自带的 iris 数据集进行处理。该数据集含有 150 个样本和 5 个变量, 其中前 4 个变量是数值变量, 最后一个变量是每个样本的类别信息。在下面的 R 程序中, 我们首先使用 data 函数来导入数据并只考虑前 4 个数值变量, 然后利用数据的前 100 个样本计算每个变量的均值 μ_j 和标准差 σ_j 并标准化, 之后再使用计算出的 μ_j 和 σ_j 对后面的 50 个样本进行同样的标准化:

```
data(iris)
D <- iris[,1:4]
D1 <- D[1:100,]
D2 <- D[101:150,]
D1_normalized <- scale(D1, center = T, scale = T)
D2_normalized <- scale(D2,
                        center = attr(D1_normalized, 'scaled:center'),
                        scale = attr(D1_normalized, 'scaled:scale'))
```

在使用 R 中的 scale 函数时, 有两个重要参数 center 和 scale。当 center 和 scale 是逻辑型值时, 分别表示在进行数据标准化时是否减去均值和除以标准差。当 center 和 scale 是数值型的向量时, 表示在进行标准化时不需要计算数据的均值和标准差, 直接调用 center 和 scale 所赋予的值即可。在上面的例子中, 前一个标准化过程将 center 和 scale 设为逻辑型值, 后一个标准化过程使用前面数据的均值和标准差进行数据的标准化。这里的 attr(D1_normalized, 'scaled:center') 和 attr(D1_normalized, 'scaled:scale') 都是向量, 在 R 中可以直接打印出来:

```
> attr(D1_normalized, 'scaled:center')
Sepal.Length Sepal.Width Petal.Length Petal.Width
      5.471      3.099      2.861      0.786
> attr(D1_normalized, 'scaled:scale')
Sepal.Length Sepal.Width Petal.Length Petal.Width
 0.6416983    0.4787389    1.4495485    0.5651531
```

我们也可以使用 apply 函数来手动验证 D1_normalized 中的每列都是均值为 0、标准差为 1。具体代码和结果如下:

```
> apply(D1_normalized, 2, sum)
Sepal.Length Sepal.Width Petal.Length Petal.Width
-1.497413e-14 -4.233159e-14 -1.454392e-14 -1.032507e-14
> apply(D1_normalized, 2, sd)
Sepal.Length Sepal.Width Petal.Length Petal.Width
1 1 1 1
```

4.3.3 删除已有变量

在很多实际数据中,经常存在冗余甚至重复数据。在这种情况下,我们需要删除一些冗余和重复的变量。一方面,删除这些变量之后能够降低数据的规模,进而降低算法的计算时间;另一方面,有些算法在存在冗余数据时会导致性能降低。以线性回归为例,在极端的情况下,如有两个变量 $v_1 = -v_2$,则必须使用正则化项,否则性能会严重降低。关于线性回归的具体讨论参见第5章。

为了删除冗余变量,我们可以采用主成分分析(principal component analysis, PCA)来进行降维。主成分分析的原理和使用参见3.3.5节和3.3.6节。

使用主成分分析的缺点是新变量是原来变量的线性组合,这样一般难以解释新变量。因此,我们一般采用一些启发式方法(heuristic methods)来删除那些冗余甚至重复的变量。首先,我们可以计算变量两两之间的相关系数。若相关系数接近1或者-1,则说明对应的这两个变量之间存在线性相关性,我们需要删除其中一个变量。

在实际操作中,为了消除变量之间的线性相关性,我们可以要求任何两个变量之间的相关系数的绝对值低于一个阈值(如0.75)。虽然这种方法只考虑了两两之间的相关系数而忽视了多个变量之间的相互关系,但是在很多情况下这种简单的处理方法也能取得较好的效果。这里我们简单介绍一种启发式方法来控制变量两两之间相关系数的最大值,如算法4-1所示。

算法4-1 通过相关系数删除冗余数据

(1) 计算变量两两之间的相关系数,得到一个 $d \times d$ 的矩阵。若该矩阵中所有元素的绝对值都小于规定的阈值 τ ,则退出。

(2) 从该矩阵中选出相关系数绝对值最大的两个变量(记为 v_1 和 v_2)。

(3) 计算变量 v_1 和所有其他变量的相关系数的绝对值平均值,记为 c_1 ; 同样,为变量 v_2 计算对应的 c_2 。

(4) 如果 $c_1 \geq c_2$,则删除变量 v_1 ; 否则删除变量 v_2 。

(5) 重复步骤(2)~(4),直到剩余变量两两之间相关系数的绝对值都小于规定的阈值。

此外,我们需要着重指出:如果一个变量在数据集中只有一个值,则意味着该变量没有为模型的建立提供任何有用的信息。一般而言,可以直接删除该变量。

4.3.4 数据的变换

在很多实际问题中,直接使用原始数据进行建模很难达到令人满意的性能。很多时候,我们需要对数据进行变换,生成若干新的特征。例如,我们可以计算两个变量的比值作为一个新的变量。假设我们的任务是要根据一组数据判定每种材料是塑料还是金属。在原始数据中,我们给定了 v_1 是质量、 v_2 是体积,则计算比值 v_1/v_2 是每种材料对应的密度。通过增加新的变量,该分类问题可以得到很好的解决。

在很多与时间相关的问题中,我们也可以从原始数据中计算很多关于时间的新特征。假设我们要预测淘宝上某个商家在11月每天的销售额。我们知道,11月11日一般是淘宝集中促销的日子,消费者在11月的消费模式与11月11日有很大关系:在11月11日之前,有很多的消费者会持币待购,而在11月11日之后则可能会敞开购买。因此,我们可以构建如下变量。

(1) 当前日期是否在11月11日之后,若是,则为1,否则为0。

(2) 计算当前日期与11月11日所隔天数的绝对值。例如,当前日期是11月9日,则为2;当前日期为11月13日,也为2。

同时我们也要注意,当前日期是否为周末对于销售量也有较大影响,因此,我们还可以计算以下变量:

(3) 检查当前日期是否为周末,若是,则为1,否则为0。

从上面的这两个例子可以看出,如何建立合理的新变量,与所要解决的问题息息相关。此外,不同的模型对于特征的要求也不一样。

下面我们给出一些常用的数据变换函数:

- x^k , 包括 \sqrt{x} 、 $\frac{1}{x}$ 等;
- $\log x$
- $\exp(x)$
- $|x|$
- $\sin x$ 和 $\cos x$
- $\frac{1}{x}$

上面列出的变换函数都是适用于单个原始变量的变化函数。相应地,我们也可以同时对多个原始变量进行变换以得到新的变量,如 x_1/x_2 等。

4.3.5 构建新的变量:哑变量

在建模中,不是所有的模型或者算法都能够直接处理分类变量(categorical variable)。有些模型,如基于决策树的模型,能够较好地处理分类变量。但另外一些模型,如线性回归和

逻辑回归，不能直接处理分类变量。在这种情况下，一种通用的方法是将分类变量转化为多个哑变量（dummy variable），从而能够使用那些模型。

哑变量的取值只能为 0 或者 1。在前面的数据集中，学生的性别只能为男或者女，因此，我们可以将分类变量“性别”转化为哑变量“性别=男”。因此，前面的数据可以转化为如表 4-3 所示的形式。

表 4-3 哑变量构建实例

学号	姓名	性别=男	年龄	体重	身高
201503345	李乐	1	21	65	175

这样，我们在考虑“性别=男”、年龄、体重、身高这 4 个特征时，可以直接使用线性回归或者逻辑回归等模型。

注意，在这个例子中，我们也可以构建“性别=女”作为新的变量，但没有必要同时构建“性别=男”和“性别=女”这两个哑变量。因为如果“性别=男”为 1 的话，“性别=女”肯定为 0，所以没有必要引入冗余的哑变量。另外，如果在构建哑变量的时候引入冗余信息，在有些模型如线性回归中会导致计算方面的问题，具体的讨论参见本书第 5 章。

下面再给出一个有关季节变量的实际例子。季节变量 v_0 的取值有 4 个（春、夏、秋、冬），则可引入 3 个哑变量 v_1 、 v_2 、 v_3 。其中 v_1 表示是否为春季， v_2 表示是否为夏季， v_3 表示是否为秋季。如果 v_0 是冬季，那么 $v_1 = v_2 = v_3 = 0$ 。表 4-4 显示了如何将季节变量 v_0 转化为 3 个哑变量 v_1 、 v_2 、 v_3 。

表 4-4 季节变量转化为哑变量

v_0	v_1	v_2	v_3
春	1	0	0
夏	0	1	0
秋	0	0	1
冬	0	0	0

一般来说，如果一个分类变量 x_i 有 k 种不同的取值，我们可以建立 $k-1$ 个新的哑变量来代替。假设我们把 x_i 的 k 种不同取值记为 $\{v_{i1}, v_{i2}, \dots, v_{ik}\}$ ，则可以将每个哑变量定义为“ x_i 的取值为 v_{ij} ”，这里 $j=1, 2, \dots, k-1$ 。

如果一个分类变量的不同取值太多，则需要做进一步的处理。如果直接转化为哑变量的话，会生成大量的哑变量，而且大部分哑变量的值为 0。事实上，对那些能够直接处理分类变量的模型（如决策树）来说，一个分类变量如果有过多的不同取值，也会影响这个变量在模型中的使用。

在这种情况下，一种方法是将那些取值太多的分类变量进行简化，以减少可能的取值数目。例如，如果该分类变量是顺序变量，那么可以将相邻的几个不同值归约到同一个值。这里我们用一个简单的例子加以说明。在前面的学生体检例子中，假设对于某项指标有负责检

查的医生进行打分,按照从差到好的程度给出 A、B、C、D、E 和 F。为了缩小该变量的取值范围,可以按照如下规则进行归约:

$A, B \Rightarrow \text{差}, C, D \Rightarrow \text{中}, E, F \Rightarrow \text{好}$

这样,这个变量的取值范围从 6 个值缩小到 3 个值,从而方便后续处理。

如果分类变量不是顺序变量,则需要其他的方法来缩小该变量的取值范围。通常情况下需要进一步分析该变量的具体含义及所要解决的问题而决定。例如,学生有家庭地址一栏。事实上,基本上每个学生的家庭地址都不一样。那么,“家庭地址”这个分类变量的取值就有很多种,很难在模型中直接应用。实际上,对于家庭地址这个变量,我们可以根据问题和数据的具体特点规约到不同的值。例如,如果体检的对象是小学生,由于小学生都住得比较接近,我们可以将家庭地址归约到街道名称这一级。换言之,我们由“家庭地址”构造新的变量“家庭地址街道名称”。如果体检的对象是大学生,一般而言,大学里面的学生来自于全国各地,我们将家庭地址规约到家庭地址所对应的省或者市即可。具体是省还是市,则需要根据所要解决的问题来进一步决定。

4.3.6 离群数据的处理

在实际数据中,我们经常会碰到离群数据(outlier)。通常,我们将那些离主流数据很远的的数据点定义为离群数据。在合适的假设下,我们可以用严格的统计语言来描述离群数据。但在实际中,很难对离群数据给出严格的定义。更多的时候,我们可以通过图/表等工具来探索数据,并识别出离群数据。同时,在识别离群数据时要非常小心,要仔细辨别离群数据是否来自于错误的观测数据等。

在建模时,有些模型对于离群数据是较为健壮的,如基于决策树的模型。因为在基于决策树的模型中,通常是通过类似于如下的规则来进行分类或者回归的:

- (1) 变量 j 的值是否大于或者小于某个阈值(对于数值变量);
- (2) 变量 j 的值是否等于某个值(对于分类变量)。

换言之,在基于决策树的模型中,我们根本没有考虑对应的变量 j 的值与主流值的差距,而只考虑它在哪一边。因此,基于决策树的模型对于离群数据的健壮性非常强。

但是,如果所采用的模型对于离群数据是较为敏感的,那么我们需要对数据进行一些预处理以消除离群数据对于模型的影响。一种常用的方法是对数据分组(binning),来对变量中的离群数据进行处理。具体而言,可将所有样本变量的取值从小到大排列好,然后分为若干组。这样,对于离群数据,我们可以使用对应组中数据的均值或者中位数等来对它进行修正。在实际中,常用的分组方法有以下两种。

- 等距分组(equal-interval binning): 将整个数据分布区间分成若干个等长的子区间。
- 等频分组(equal-frequency binning): 在划分过程中使得每个区间的样本数一样。

4.4 数据可视化

数据可视化 (data visualization) 是研究数据具体分布的有效方法。通过数据可视化, 可以获得对数据的直观认识, 为下一步的处理打下较好的基础。本节介绍如下方法:

- 直方图;
- 柱状图;
- 茎叶图;
- 箱线图;
- 散点图。

利用这些工具, 可以生动直观地了解数据的具体分布, 为数据的后续处理和建模工作提供很大的便利。在本节中, 除了介绍这些工具的基本原理之外, 还介绍 R 中对应的函数, 这样读者可以在实际中方便地作图以分析、探索数据。

在本章中我们主要介绍 R 中提供的基本的画图函数。在 R 中有一个非常流行的包叫 ggplot2。本节讨论的所有数据探索方法都可以同时利用 R 中的基本画图函数和 ggplot2^[23]来实现。简单起见, 我们主要介绍 R 中的基本函数。本节所有的程序都在文件 R_data_visualization.R 中。

4.4.1 直方图

直方图 (histogram) 是显示样本分布的有效方法。在直方图中, 我们将数据的取值区间划分为若干个子区间, 并计算整个样本集合中落入每个子区间的样本数。在作图时, 为每个子区间画一个矩形, 它的高度与落入子区间的样本数相对应。

在画直方图时, 子区间的数目对直方图的最终形态有很大影响。子区间数目太少, 直方图所反映的概率密度的形态就不是很准确; 子区间数目太多, 由于随机性的影响, 相邻子区间的数目分布可以差异很大。因此, 在画直方图时, 我们需要选择合适的子区间数目, 使得直方图能够较为真实地反映总体的概率分布。若所划分的子区间间隔相等, 则每个子区间的长度称为组距。

在 R 中, 我们可以使用函数 hist 来画直方图。在 hist 函数中, 我们可以使用 breaks 参数来指定子区间的数目。在本节给出的关于 R 的例子中, 我们使用 iris 数据和 mtcars 数据来讲述如何在 R 中快速地探索数据。由于这两组数据是 R 中自带的, 因此我们可以使用 data 函数来直接载入它们:

```
# Step 1. Load the data
data("iris")
data("mtcars")
```

之后, 我们可以直接使用两个数据框: iris 和 mtcars。在下面的代码中, 我们使用 hist 函数画出了多个直方图。

```

# Step 2. Plot the histogram using mtcars$mpg
# Plot a basic histogram
hist(mtcars$mpg)
# Plot a histogram with 10 bins
# Specify approximate number of bins with breaks
hist(mtcars$mpg, breaks=10)
# Plot a red histogram with 12 bins
hist(mtcars$mpg, breaks=12, col="red")
# Add labels
hist(mtcars$mpg, breaks=12, col="red", xlab = "Miles Per Gallon", main =
"Histogram of mpg")

```

在第一个例子中,我们指定数据 `mtcars$mpg`, 并使用 `hist` 函数中的默认参数来画直方图, 结果如图 4-1 所示。在第二个例子中,我们虽然仍使用 `mtcars$mpg` 数据,但是通过设置 `breaks` 参数指定子区间的数目为 10, 其结果如图 4-2 所示。注意,这里指定的子区间的数目是一个大概数目, `hist` 函数会根据数据的具体特征进行适当的调整,在图 4-2 中我们就得到了 12 个子区间。在第三个例子中,我们将子区间的数目指定为 12, 并将直方图的颜色设置为红色, 结果如图 4-3 所示。在最后一个例子中,我们在第三个例子的基础上,将 x 轴的标注指定为 Miles Per Gallon, 将整个直方图的标题指定为 Histogram of mpg, 结果如图 4-4 所示。

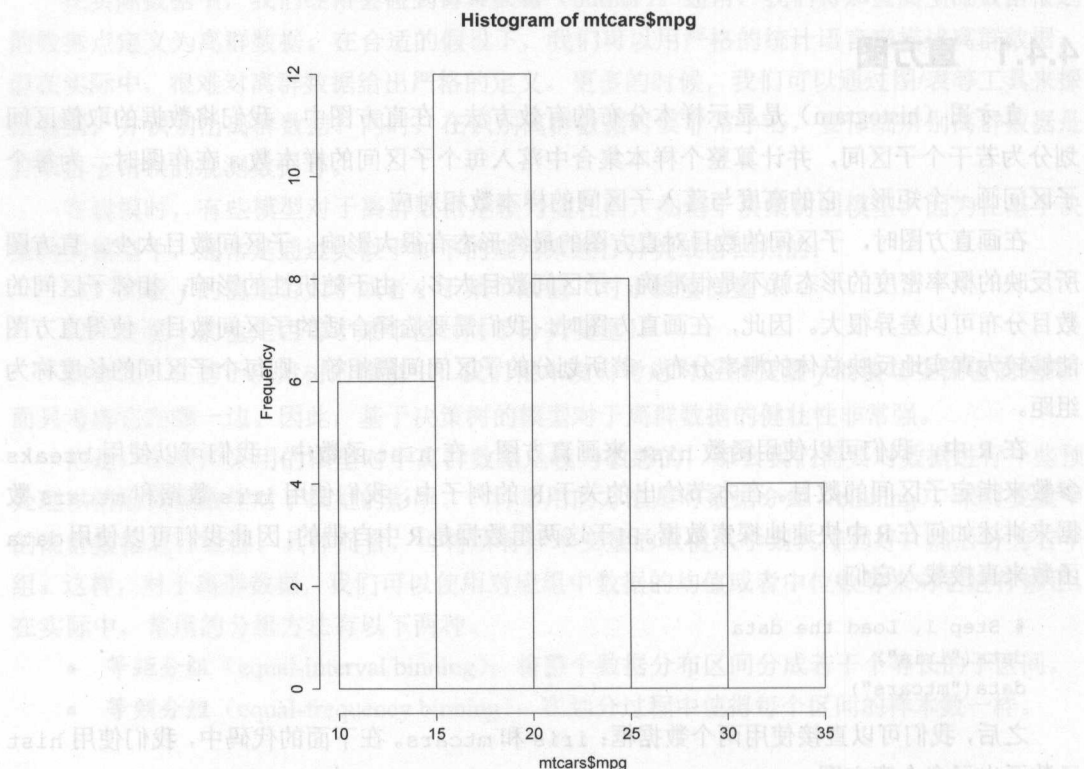


图 4-1 mtcars 数据中 mpg 列对应的直方图 (子区间数目为 5)

Histogram of mtcars\$mpg

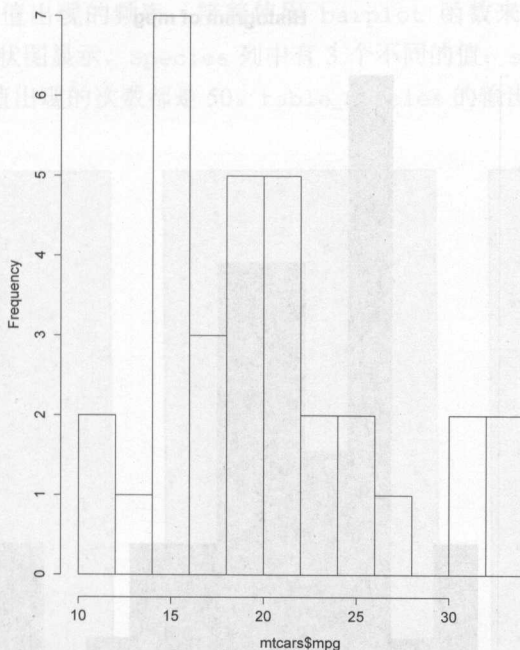


图 4-2 mtcars 数据中 mpg 列对应的直方图（子区间数目为 12）

Histogram of mtcars\$mpg

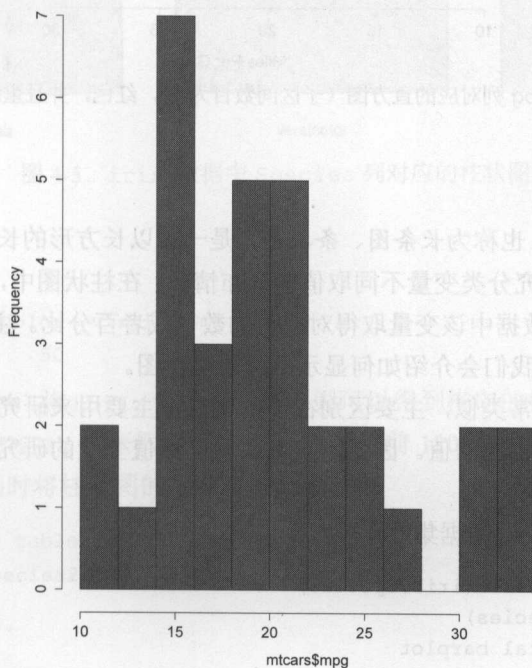


图 4-3 mtcars 数据中 mpg 列对应的直方图（子区间数目为 12，颜色为红色）

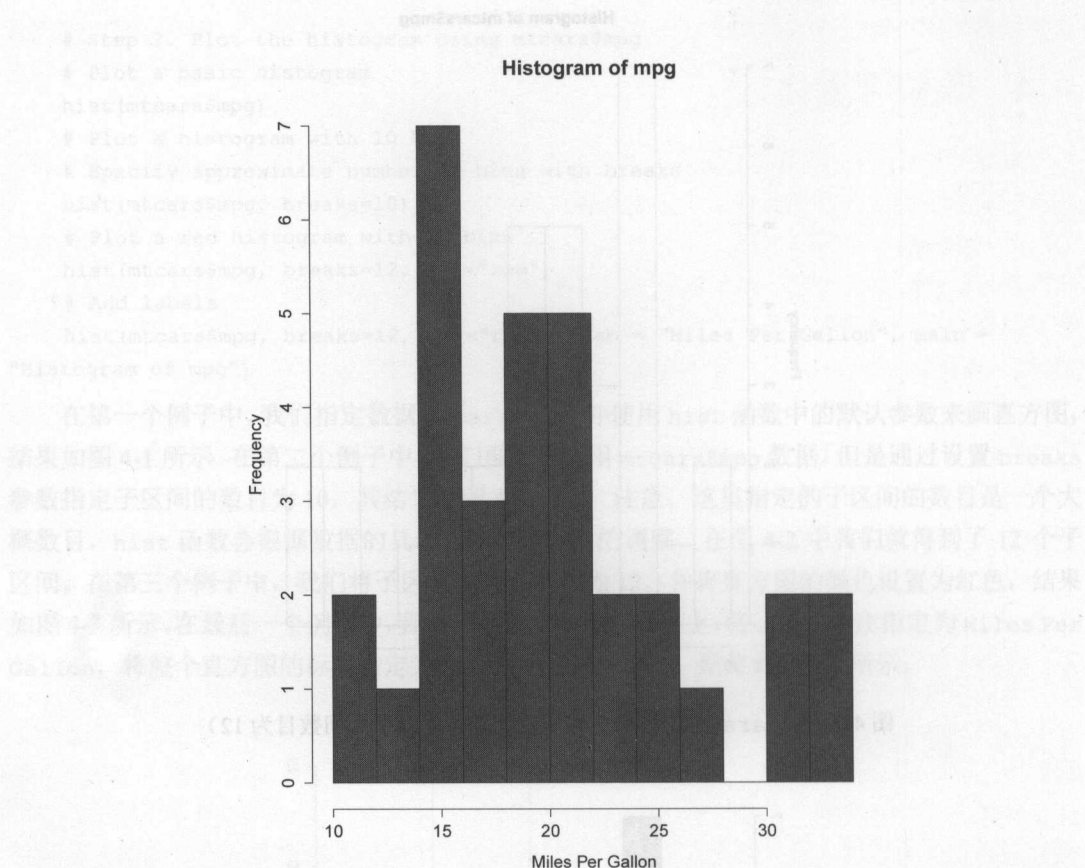


图 4-4 mtcars 数据中 mpg 列对应的直方图 (子区间数目为 12, 红色, 并且重新指定 x 轴的标注和标题)

4.4.2 柱状图

柱状图 (bar chart) 也称为长条图、条状图, 是一种以长方形的长度表示变量信息的统计图。柱状图通常用来研究分类变量不同取值的分布情况。在柱状图中, x 轴是分类变量所取的不同值, y 轴是样本数据中该变量取得对应值的数目或者百分比。注意, 柱状图也可以横向表示, 在下面的例子中我们会介绍如何显示不同的柱状图。

柱状图与直方图非常类似, 主要区别在于: 直方图主要用来研究数据的分布, 而柱状图主要用来比较某个变量不同的值。因此, 直方图用于数值变量的研究和探索, 而柱状图用于分类变量的探索。

我们在这里考察 iris 数据集:

```
table_Species = table(iris$Species)
barplot(table_Species)
# plot a horizontal barplot
barplot(table_Species, horiz=T)
```

在这个例子中，我们首先使用 `table` 函数来计算 `iris` 数据框中 `Species` 列中有多少个不同的值，以及每个值出现的频率，接着使用 `barplot` 函数来显示对应的柱状图（见图 4-5）。根据生成的柱状图显示，`Species` 列中有 3 个不同的值：`setosa`、`versicolor` 和 `virginica`，并且每个值出现的次数都是 50。`table_Species` 的输出如下：

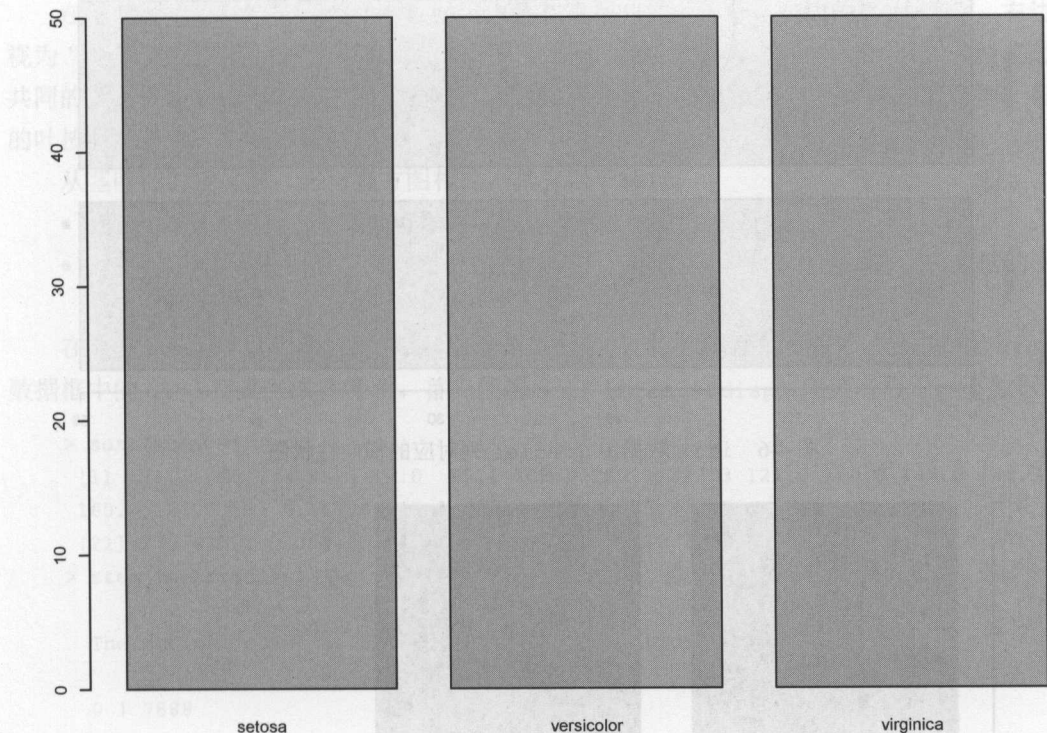


图 4-5 `iris` 数据中 `Species` 列对应的柱状图

```
> table_Species
```

```
setosa versicolor virginica
    50         50         50
```

在 `barplot` 函数中，将 `horiz` 参数设为真，则可以得到横向的柱状图，如图 4-6 所示。

我们还可以只考虑 `iris` 数据框中 `Species` 列中的前 120 行，并输出对应的柱状图。在下面的例子中，我们同时将柱状图的颜色设为蓝色：

```
table_Species2 = table(iris[1:120, "Species"])
barplot(table_Species2, col='blue')
```

图 4-7 是输出的柱状图。

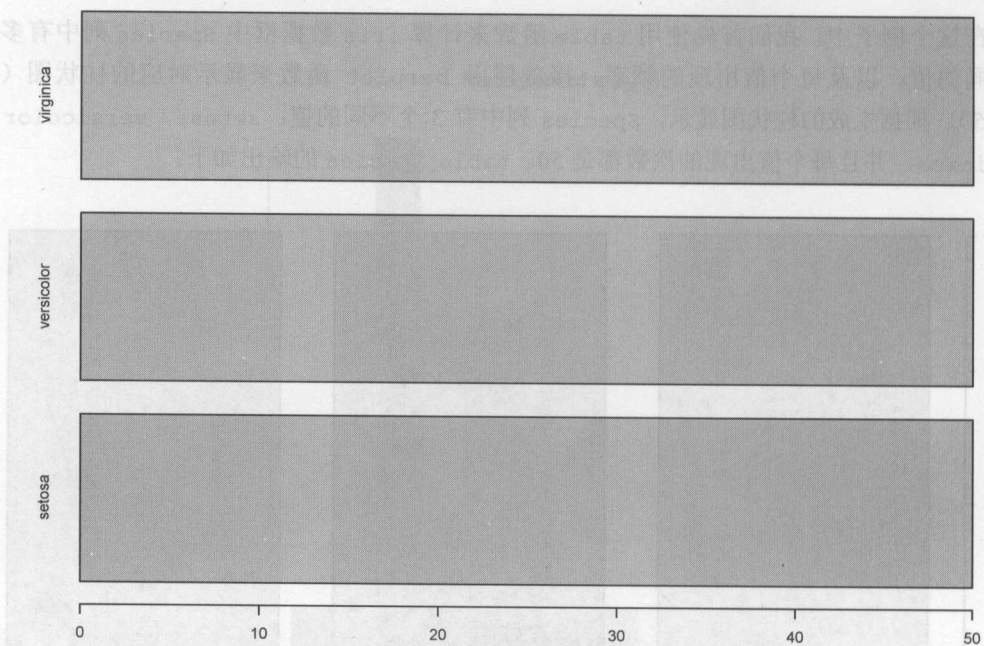


图 4-6 iris 数据中 Species 列对应的横向柱状图

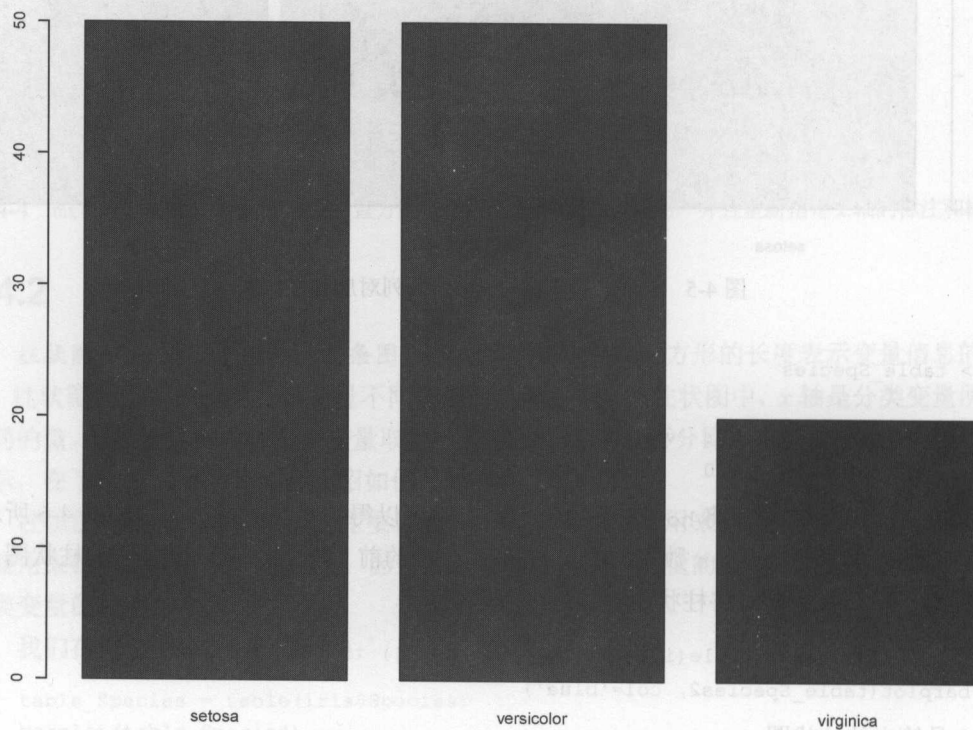


图 4-7 iris 数据中 Species 列中前 120 行对应的柱状图 (蓝色)

4.4.3 茎叶图

与直方图相比，茎叶图（stem-and-leaf plot）能够更加细致地观察样本数据的分布，因此也是探索数据分布的有效方法之一。

在茎叶图中，我们把每个样本数据用一条竖线划分为两部分，左边的视为“茎”，右边的视为“叶”。我们把所有样本中具有相同茎的数据点排成一横行，并用竖线隔开。竖线左边是共同的茎，竖线右边是所有的叶（去除茎之后的部分）。注意，在茎叶图中，我们将同一行中的叶按照从小到大的顺序排好。

从茎叶图的定义看，它与直方图相比，具有以下特点。

- 茎叶图和直方图一样，都可以直观地观察数据的分布情况。
- 利用茎叶图，可以很自然地对数据进行排序。因此，可以简单地得到这组数据的次序统计量。

在 R 中，我们可以简单利用 stem 函数来得到茎叶图。在下面的例子中，我们使用 mtcars 数据框中的 disp 列来生成茎叶图。首先使用 sort (mtcars\$disp) 向读者展示一下数据。

```
> sort(mtcars$disp)
[1] 71.1 75.7 78.7 79.0 95.1 108.0 120.1 120.3 121.0 140.8 145.0 146.7
160.0 160.0 167.6 167.6 225.0 258.0 275.8 275.8
[21] 275.8 301.0 304.0 318.0 350.0 351.0 360.0 360.0 400.0 440.0 460.0 472.0
> stem(mtcars$disp)
```

The decimal point is 2 digit(s) to the right of the |

```
0 | 7888
1 | 012224
1 | 556677
2 | 3
2 | 6888
3 | 002
3 | 5566
4 | 04
4 | 67
```

接下来我们直接使用 stem 函数生成茎叶图。注意，R 的输出中提示小数点在竖线的右边两位处。因此，我们考察第一行，茎是 0，意味着这一行对应的数据是 0**.**的形式。根据 sort (mtcars\$disp) 的结果，我们知道有 71.1、75.7、78.7、79.0、95.1。如果在叶部分只用一位有效数字来表示这几个样本，则有

$$71.1 \approx 0.7 \times 10^2$$

$$75.7 \approx 0.8 \times 10^2$$

$$78.7 \approx 0.8 \times 10^2$$

$$79.0 \approx 0.8 \times 10^2$$

$$95.1 \approx 1.0 \times 10^2$$

注意，四舍五入之后 95.1 已经不满足要求了，需要将它放入下一行处理。因此，0 这一行对应的叶为 7、8、8、8，如上面的输出所示。

对于规模比较小的数据，茎叶图能够得到比较好的效果。当数据规模较大时，可以先对数据取样 (sampling)，再使用茎叶图研究数据。

4.4.4 箱线图

箱线图 (box plot) 也是一种探索数据的有效工具。与茎叶图相比，箱线图更加简洁直观。在箱线图中，我们利用如下统计量来画图从而反映数据的分布：

- 下四分位数；
- 上四分位数；
- 中位数。

画箱线图时，其基本原理是先画一个矩形，矩形的两端分别为下四分位数和上四分位数，而在矩形中间有一条线代表中位数；然后，从矩形的两端各画一条虚线到达不是异常值的最大/最小值。

在 R 中，我们可以直接使用 `boxplot` 函数生成箱线图。以 `mtcars` 数据框为例，我们从 `boxplot` 最简单的用法开始讨论，并介绍涉及多个变量时如何生成箱线图。

```
# Plot a basic boxplot using only 1 variable
boxplot(mtcars$mpg)
boxplot(mtcars$mpg, horizontal=T)
# Plot a basic boxplot for cars whose cyl=4
boxplot(mtcars[mtcars$cyl==4, 'mpg'])
# Plot a box plot involving 2 variables.
boxplot(mpg~cyl, data=mtcars, main="Car Milage Data",
        xlab="Number of Cylinders", ylab="Miles Per Gallon")
# Plot box plot involving 3 variables
boxplot(mpg~cyl*gear, data=mtcars,
        col=c("blue", "red")),
        main="Car Milage Data", xlab="Cylinder and Gear")
```

在这段 R 程序中，我们首先直接使用 `boxplot` 函数生成 `mtcars` 数据框中 `mpg` 列对应的箱线图，如图 4-8 所示。

我们也可以生成水平的箱线图，只需要将参数 `horizontal` 设为真即可。`mtcars` 数据框中 `mpg` 列对应的横向箱线图如图 4-9 所示。

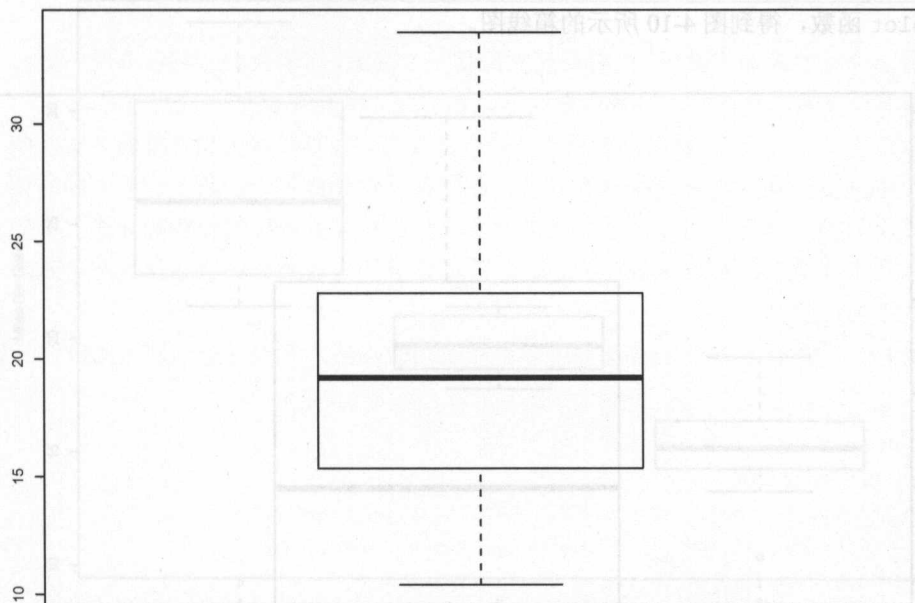


图 4-8 mtcars 数据中 mpg 列对应的箱线图

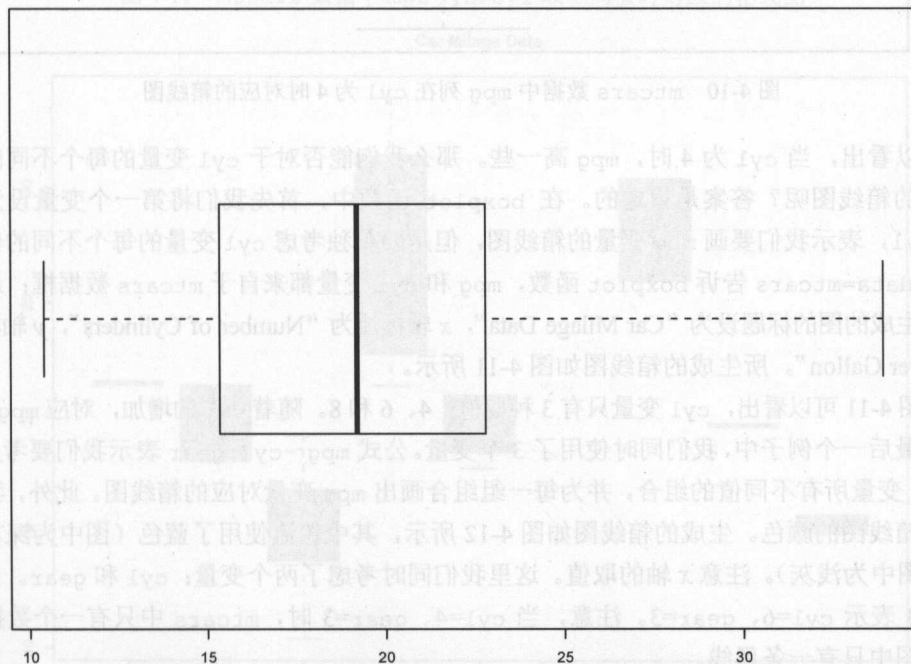


图 4-9 mtcars 数据中 mpg 列对应的横向箱线图

接下来我们讨论涉及多个变量的箱线图。首先只考虑 mtcars 数据框中当变量 cyl 为 4

时 `mpg` 列的分布。我们可以使用 `mtcars[mtcars$cyl==4, 'mpg']` 得到对应的数据，直接调用 `boxplot` 函数，得到图 4-10 所示的箱线图。

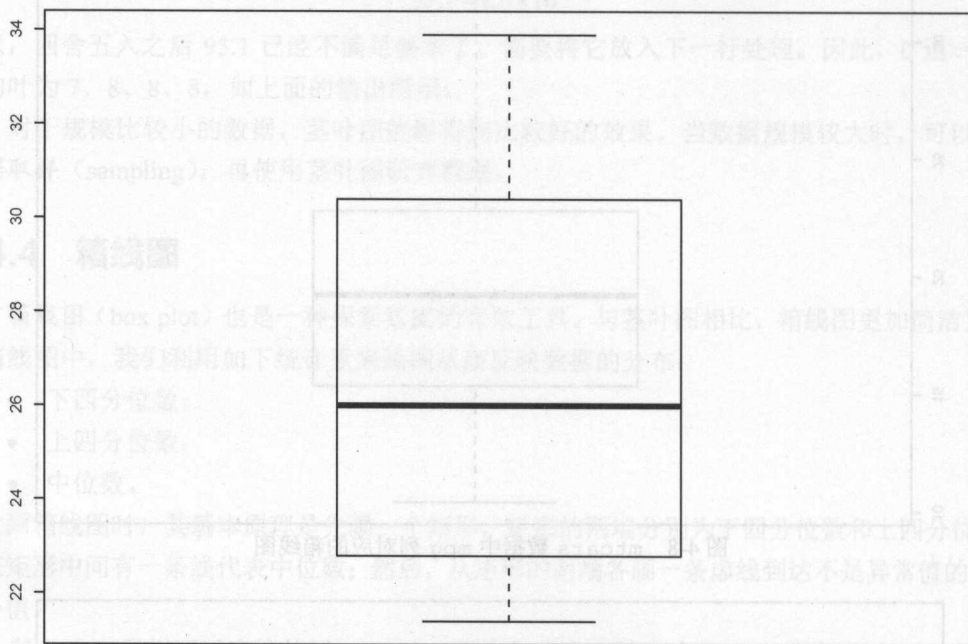


图 4-10 mtcars 数据中 mpg 列在 cyl 为 4 时对应的箱线图

可以看出，当 `cyl` 为 4 时，`mpg` 高一些。那么我们能否对于 `cyl` 变量的每个不同的值画出相应的箱线图呢？答案是肯定的。在 `boxplot` 函数中，首先我们将第一个变量设为公式 `mpg~cyl`，表示我们要画 `mpg` 变量的箱线图，但是要单独考虑 `cyl` 变量的每个不同的值；然后使用 `data=mtcars` 告诉 `boxplot` 函数，`mpg` 和 `cyl` 变量都来自于 `mtcars` 数据框；最后我们将所生成的图的标题设为“Car Milage Data”，`x` 轴标注为“Number of Cylinders”，`y` 轴标注为“Miles Per Gallon”。所生成的箱线图如图 4-11 所示。

从图 4-11 可以看出，`cyl` 变量只有 3 种取值：4、6 和 8。随着 `cyl` 的增加，对应 `mpg` 变小。

在最后一个例子中，我们同时使用了 3 个变量。公式 `mpg~cyl*gear` 表示我们要考虑 `cyl` 和 `gear` 变量所有不同值的组合，并为每一组组合画出 `mpg` 变量对应的箱线图。此外，我们还指定了箱线图的颜色。生成的箱线图如图 4-12 所示，其中轮流使用了蓝色（图中为深灰）和红色（图中为浅灰）。注意 `x` 轴的取值。这里我们同时考虑了两个变量：`cyl` 和 `gear`。`x` 轴上面的 6.3 表示 `cyl=6`、`gear=3`。注意，当 `cyl=4`、`gear=3` 时，`mtcars` 中只有一个数据点，因此在图中只有一条黑线。

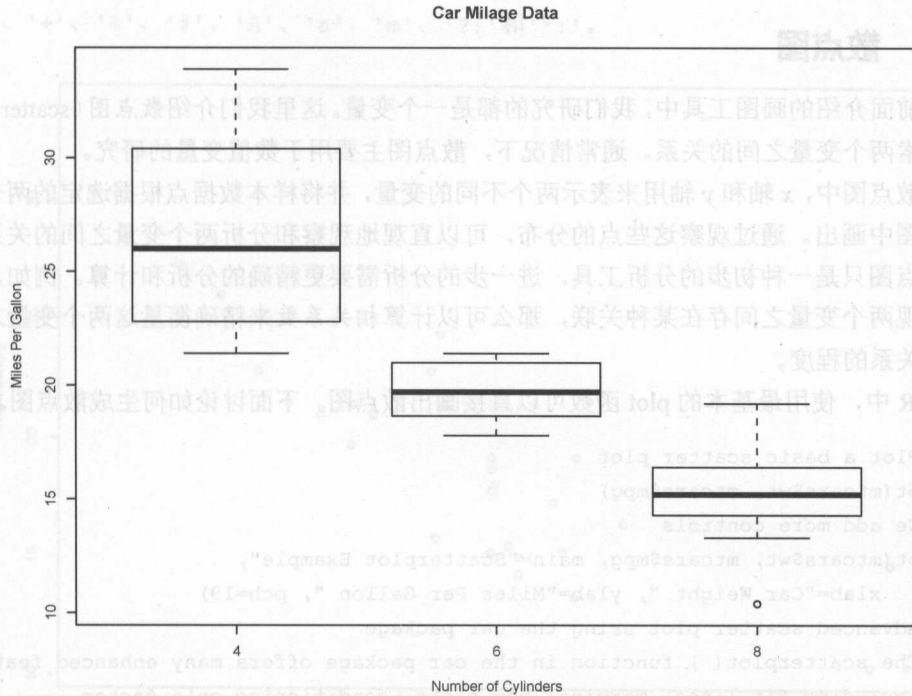


图 4-11 mtcars 数据中 mpg 列在 cyl 取不同值时对应的箱线图

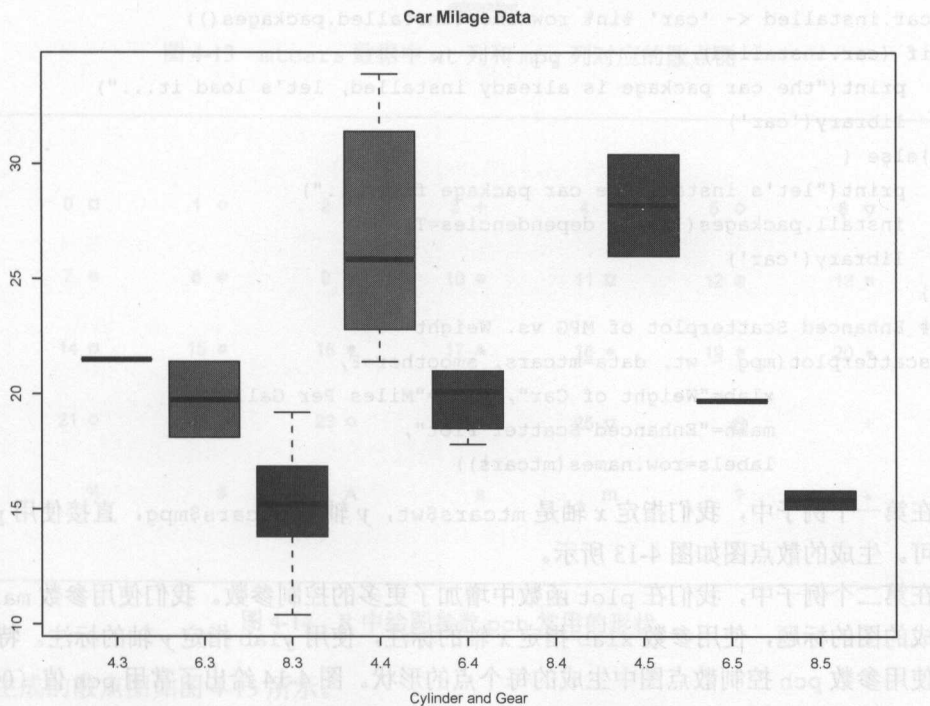


图 4-12 mtcars 数据中 mpg 列在 cyl 和 gear 取不同值时对应的箱线图

4.4.5 散点图

在前面介绍的画图工具中,我们研究的都是一个变量。这里我们介绍散点图(scatter plot),用来探索两个变量之间的关系。通常情况下,散点图主要用于数值变量的研究。

在散点图中, x 轴和 y 轴用来表示两个不同的变量,并将样本数据点根据选定的两个变量的值在图中画出。通过观察这些点的分布,可以直观地观察和分析两个变量之间的关系。注意,散点图只是一种初步的分析工具,进一步的分析需要更精确的分析和计算。例如,如果我们发现两个变量之间存在某种关联,那么可以计算相关系数来精确衡量这两个变量之间存在线性关系的程度。

在R中,使用最基本的plot函数可以直接画出散点图。下面讨论如何生成散点图。

```
# Plot a basic scatter plot
plot(mtcars$wt, mtcars$mpg)

# We add more controls
plot(mtcars$wt, mtcars$mpg, main="Scatterplot Example",
      xlab="Car Weight ", ylab="Miles Per Gallon ", pch=19)

# Advanced scatter plot using the car package
# The scatterplot() function in the car package offers many enhanced features,
# including fit lines, marginal box plots, conditioning on a factor,
# and interactive point identification. Each of these features is optional.
car.installed <- 'car' %in% rownames(installed.packages())
if (car.installed) {
  print("the car package is already installed, let's load it...")
  library('car')
} else {
  print("let's install the car package first...")
  install.packages('car', dependencies=T)
  library('car')
}

# Enhanced Scatterplot of MPG vs. Weight
scatterplot(mpg ~ wt, data=mtcars, smoother=F,
            xlab="Weight of Car", ylab="Miles Per Gallon",
            main="Enhanced Scatter Plot",
            labels=row.names(mtcars))
```

在第一个例子中,我们指定 x 轴是mtcars\$wt, y 轴是mtcars\$mpg,直接使用plot函数即可。生成的散点图如图4-13所示。

在第二个例子中,我们在plot函数中增加了更多的控制参数。我们使用参数main指定所生成的图的标题,使用参数xlab指定 x 轴的标注,使用ylab指定 y 轴的标注。特别地,我们使用参数pch控制散点图中生成的每个点的形状。图4-14给出了常用pch值(0~25的整数)所对应的形状。图4-14中倒数第二行最后两个形状和最后一行所有形状对应的pch值

为 '@'、'+'、'%'、'#'、'A'、'a'、'm'、'?':'' 和 '*'。

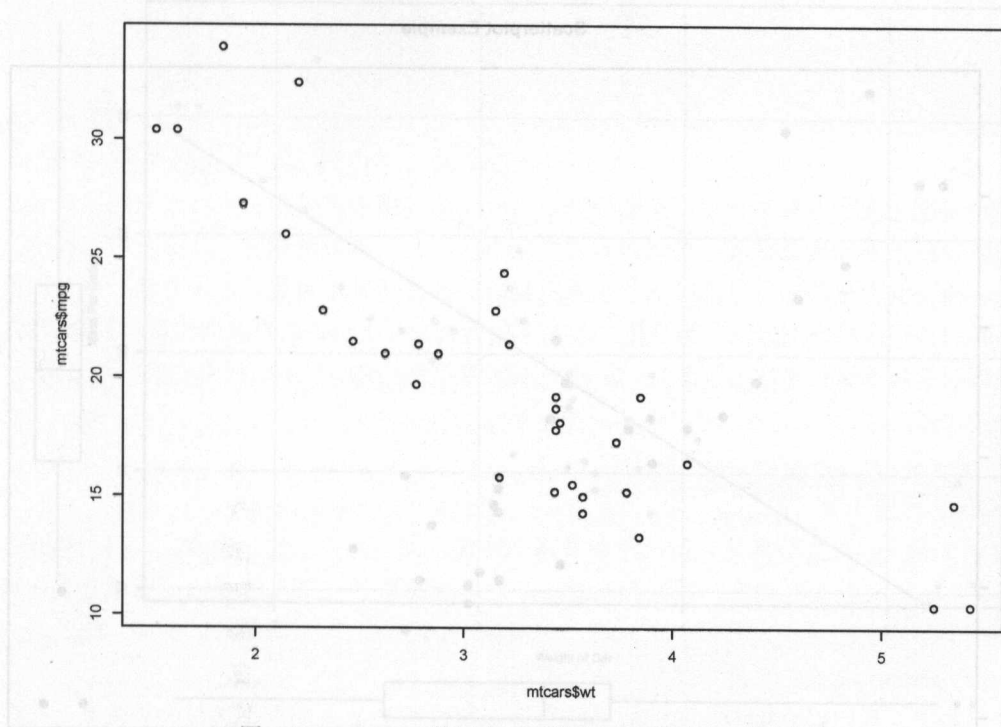


图 4-13 mtcars 数据中 wt 列和 mpg 列对应的散点图

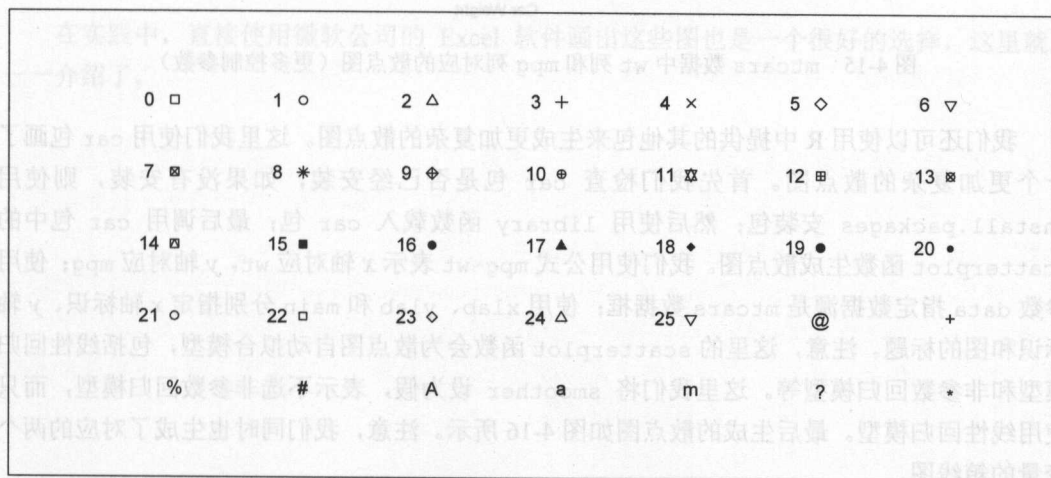


图 4-14 R 中绘图参数 pch 常用的形状

所生成的散点图如图 4-15 所示。

4.4.5 散点图

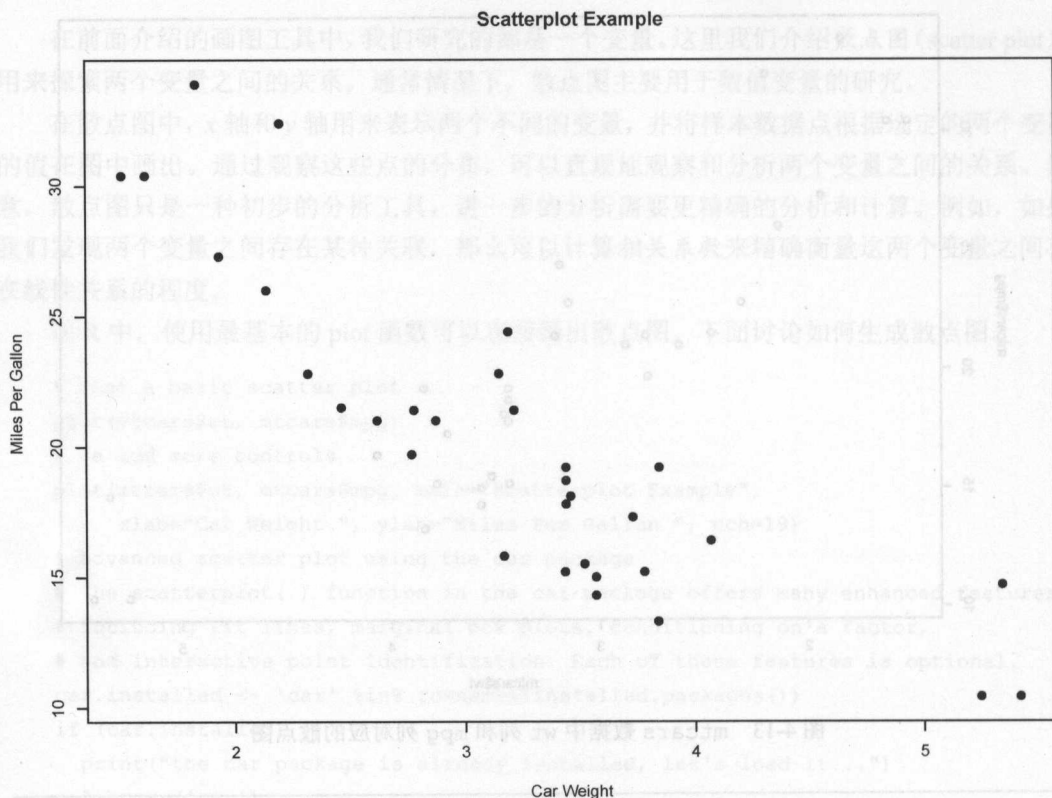


图 4-15 mtcars 数据中 wt 列和 mpg 列对应的散点图（更多控制参数）

我们还可以使用 R 中提供的其他包来生成更加复杂的散点图。这里我们使用 car 包画了一个更加复杂的散点图。首先我们检查 car 包是否已经安装，如果没有安装，则使用 `install.packages` 安装包；然后使用 `library` 函数载入 car 包；最后调用 car 包中的 `scatterplot` 函数生成散点图。我们使用公式 `mpg~wt` 表示 x 轴对应 wt，y 轴对应 mpg；使用参数 `data` 指定数据源是 mtcars 数据框；使用 `xlab`、`ylab` 和 `main` 分别指定 x 轴标识、y 轴标识和图的标题。注意，这里的 `scatterplot` 函数会为散点图自动拟合模型，包括线性回归模型和非参数回归模型等。这里我们将 `smoother` 设为假，表示不选非参数回归模型，而只使用线性回归模型。最后生成的散点图如图 4-16 所示。注意，我们同时也生成了对应的两个变量的箱线图。

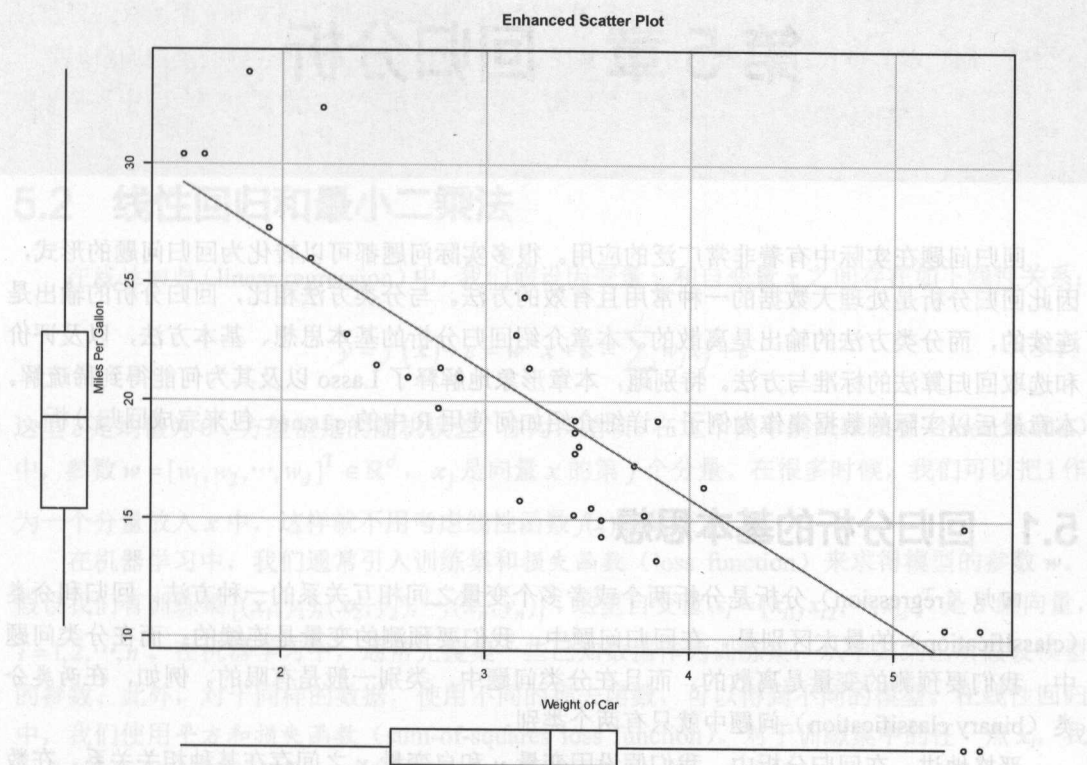


图 4-16 使用 car 包生成 mtcars 数据中 wt 列和 mpg 列对应的散点图

在实践中，直接使用微软公司的 Excel 软件画出这些图也是一个很好的选择，这里就不一一介绍了。

第5章 回归分析

回归问题在实际中有着非常广泛的应用。很多实际问题都可以转化为回归问题的形式，因此回归分析是处理大数据的一种常用且有效的方法。与分类方法相比，回归分析的输出是连续的，而分类方法的输出是离散的。本章介绍回归分析的基本思想、基本方法，以及评价和选取回归算法的标准与方法。特别地，本章形象地解释了 Lasso 以及其为何能得到稀疏解。本章最后以实际的数据集作为例子，详细介绍如何使用 R 中的 glmnet 包来完成回归分析。

5.1 回归分析的基本思想

回归 (regression) 分析是分析两个或者多个变量之间相互关系的一种方法。回归和分类 (classification) 的最大区别是，在回归问题中，我们要预测的变量是连续的；而在分类问题中，我们要预测的变量是离散的。而且在分类问题中，类别一般是有限的。例如，在两类分类 (binary classification) 问题中就只有两个类别。

严格地讲，在回归分析中，我们假设因变量 y 和自变量 \mathbf{x} 之间存在某种相关关系。在数据分析中，假设有一个数据集：

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$$

这里 $\mathbf{x}_i \in \mathbb{R}^d$ 是 $d(d \geq 1)$ 维向量， $y_i \in \mathbb{R}$ 。我们的目的是根据该数据集推断出函数 $f(\mathbf{x})$ ：

$$y = f(\mathbf{x})$$

这里，我们将 y 称为因变量，将 \mathbf{x} 称为自变量。函数 $f(\mathbf{x})$ 称为 y 对 \mathbf{x} 的回归函数。如果 $d=1$ ，则称为一元回归分析，因为只有一个自变量；如果 $d>1$ ，则称为多元回归分析。按照函数 f 的类型，回归分析可分为线性回归分析和非线性回归分析。

“回归”一词最早由法兰西斯·高尔顿 (Francis Galton) 引入。他研究了子女身高和父母身高的关系。当时他发现虽然父母的身高可以影响子女的身高，但是子女的身高却有逐渐“回归”到中等的现象。现在回归一词的含义已经发生了很大变化。在现代，回归主要指的是研究 y 和 \mathbf{x} 之间的关系，其中 y 是连续型的变量。

回归分析的运用十分广泛。一些常用例子包括：

- 收入水平 (y) 与受教育程度 (x) 之间的关系；
- 子女身高 (y) 与父亲身高 (x_1)、母亲身高 (x_2) 之间的关系；
- 车辆每公里耗油量 (y) 与发动机排量 (x_1)、发动机马力 (x_2) 和车辆重量 (x_3) 之间的关系。

这里我们分别用 y 和 x 标出对应问题中的因变量和自变量。

线性回归是回归分析中最简单的例子。下面首先从最简单的线性分析开始介绍，并引入机器学习的一些重要概念，包括损失函数、模型复杂度等。

5.2 线性回归和最小二乘法

在线性回归 (linear regression) 中，我们假设因变量 y 和自变量 x 之间存在如下线性关系：

$$y = f(x) + \varepsilon = \mathbf{w}^T \mathbf{x} + \varepsilon = \sum_{j=1}^d w_j x_j + \varepsilon \quad (5-1)$$

这里 ε 是均值为 0、方差恒定的随机误差，称为误差项。在这个简单的线性模型 (linear model) 中，参数 $\mathbf{w} = [w_1, w_2, \dots, w_d]^T \in \mathbb{R}^d$ ， x_j 是向量 \mathbf{x} 的第 j 个分量。在很多时候，我们可以把 1 作为一个分量放入 \mathbf{x} 中，这样就不用考虑线性函数 $f(\mathbf{x})$ 的截距了。

在机器学习中，我们通常引入训练集和损失函数 (loss function) 来求得模型的参数 \mathbf{w} 。假设我们有训练集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ ，这里自变量 $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{id}]^T$ 是 d 维向量， $i = 1, 2, \dots, n$ 。在机器学习中，通常先搜集一些已知数据作为训练集，从中归纳出所假设模型的参数。此外，对于同样的数据，使用不同的损失函数，可以得到不同的模型。在线性回归中，我们使用平方和损失函数 (sum-of-squares loss function)。对于训练集中的任一点 \mathbf{x}_i ，我们的预测值是 $f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i = \sum_{j=1}^d w_j x_{ij}$ ，则最小化如下函数：

$$L = \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2 \quad (5-2)$$

将式 (5-1) 代入，则有

$$L = \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2 = \sum_{i=1}^n \left(y_i - \sum_{j=1}^d w_j x_{ij} \right)^2 \quad (5-3)$$

写成矩阵的形式是

$$L = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 = \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} \quad (5-4)$$

这里

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n, \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}$$

在最小二乘法 (least squares) 中，我们要求出使得损失函数最小的 \mathbf{w} 。因此，现在的问

题就是找到使得式(5-3)最小化的 $\mathbf{w}=[w_1, w_2, \dots, w_d]^T \in \mathbb{R}^d$ 。事实上式(5-3)是关于 w_j 的二次函数, 要最小化 L 只需要求得 L 关于 w_j 的导数并设为 0, 即可求出使得 L 最小的 w_j 。这里我们给出 L 关于 w_k 的导数 (这里使用下标 k):

$$\frac{\partial L}{\partial w_k} = 2 \sum_{i=1}^n \left(y_i - \sum_{j=1}^d w_j x_{ij} \right) (-x_{ik}) \quad (5-5)$$

也可以将 $\frac{\partial L}{\partial w_k}$ 综合起来写成矩阵的形式:

$$\frac{\partial L}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_d} \end{bmatrix} = 2\mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{X}^T \mathbf{y} \quad (5-6)$$

令导数 $\frac{\partial L}{\partial w_k}$ 为 0, 可得方程:

$$\sum_{i=1}^n x_{ik} y_i = \sum_{j=1}^d \left(\sum_{i=1}^n x_{ij} x_{ik} \right) w_j \quad (5-7)$$

写成矩阵的形式是:

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y} \quad (5-8)$$

该方程称为正规方程 (normal equation)。如果矩阵 $\mathbf{X}^T \mathbf{X}$ 可逆, 则 \mathbf{w} 的最小二乘估计 $\hat{\mathbf{w}}$ 为:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (5-9)$$

根据训练集得到的 $\hat{\mathbf{w}} = [\hat{w}_1, \hat{w}_2, \dots, \hat{w}_d]^T \in \mathbb{R}^d$, 对于第 i 个样本 \mathbf{x}_i , 可以得到相应的预测值:

$$\hat{y}_i = \hat{\mathbf{w}}^T \mathbf{x}_i = \sum_{j=1}^d \hat{w}_j x_{ij} \quad (5-10)$$

5.2.1 最小二乘法的几何解释

在线性回归中, 我们假设所得的模型 $y = f(\mathbf{x})$ 是一个高维空间中的超平面。在图 5-1 中, 假设 $\mathbf{x}_i \in \mathbb{R}^2$ 是二维向量, 那么 $f(\mathbf{x})$ 就是一个平面。在最小二乘法中, 我们要求误差项 $\varepsilon_i = y_i - f(\mathbf{x}_i)$ 的平方和最小。事实上, $|\varepsilon_i|$ 就是真实值 y_i 到平面上的相应点 $f(\mathbf{x}_i)$ 之间的距离。

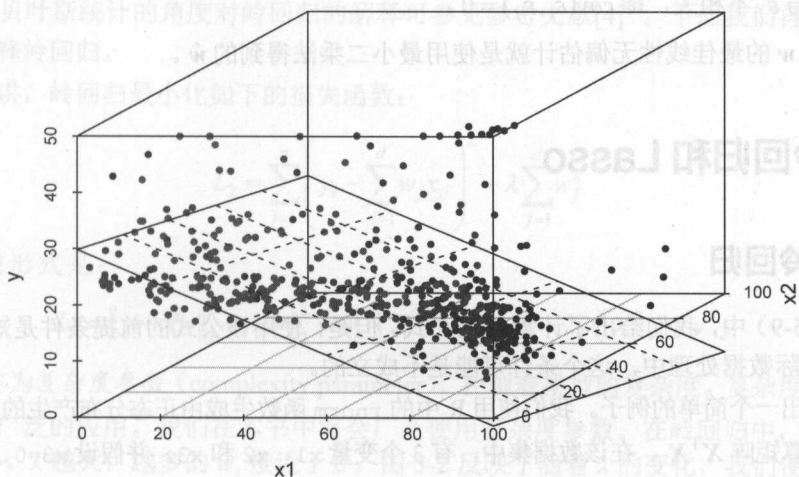


图 5-1 最小二乘法的几何解释

5.2.2 线性回归和极大似然估计

在前面的介绍中，我们从损失函数的角度出发，推导出了正规方程。事实上，我们也可以从统计学中的极大似然估计的角度推导出最小二乘法的基本公式。我们假设因变量 y_i 可以表示成如下形式：

$$y_i = \mathbf{w}^T \mathbf{x}_i + \varepsilon_i \quad (5-11)$$

这里 ε_i 是噪声，它服从均值为 0、方差为 β^{-1} 的正态分布 $N(0, \beta^{-1})$ 。由于已知 \mathbf{x}_i ，因此 y_i 满足正态分布 $N(\mathbf{w}^T \mathbf{x}_i, \beta^{-1})$ 。因此，观察到数据 $\mathbf{y} = [y_1, y_2, \dots, y_n]^T$ 的对数似然函数为：

$$\ln p(\mathbf{y} | \mathbf{w}, \beta) = \sum_{i=1}^n \ln N(y_i | \mathbf{w}^T \mathbf{x}_i, \beta^{-1}) = \frac{n}{2} \ln \beta - \frac{n}{2} \ln(2\pi) - \frac{\beta}{2} \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

可以看出，前面两项 $\frac{n}{2} \ln \beta$ 和 $\frac{n}{2} \ln(2\pi)$ 都是恒定值。最大化 $\ln p(\mathbf{y} | \mathbf{w}, \beta)$ 等价于最小化 $\sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$ 。因此，在噪声 ε_i 服从正态分布 $N(0, \beta^{-1})$ 的情况下，最大化 \mathbf{y} 的似然函数等价于最小化平方和损失函数。这也从另一个角度解释了为什么我们在回归分析中采用平方和损失函数。

高斯-马尔可夫定理 (Gauss-Markov theorem) 是统计学中非常著名的一个发现，它在理论上奠定了最小二乘法的基础。我们在这里只是简单介绍一下定理本身，而定理的具体证明从略，感兴趣的读者可以参考[2]。

在高斯-马尔可夫定理中，我们并不假设 ε_i 服从正态分布。严格地讲，我们假设 ε_i 满足：

1) ε_i 的期望值为 0，也即 $E(\varepsilon_i) = 0$ ；

2) ε_i 的方差满足 $\text{var}(\varepsilon_i) = \sigma^2 < \infty$ ；

3) ε_i 与 ε_j 不相关, 即 $\text{cov}(\varepsilon_i, \varepsilon_j) = 0$ 。

则回归系数 w 的最佳线性无偏估计就是使用最小二乘法得到的 \hat{w} 。

5.3 岭回归和 Lasso

5.3.1 岭回归

在式 (5-9) 中, 我们给出了计算 w 的公式。但是, 使用该公式的前提条件是矩阵 $X^T X$ 可逆。而在实际数据处理中, 这个条件可能是不成立的。

下面给出一个简单的例子。我们使用 R 中的 `rnorm` 函数生成由正态分布产生的 1000 个三维向量, 并计算矩阵 $X^T X$ 。在该数据集中, 有 3 个变量 x_1 、 x_2 和 x_3 , 并假设 $x_3 = 0.6x_1 + 0.4x_2$ 。相应的 R 代码如下:

```
set.seed(10)
n = 1000
x1 = rnorm(1000)
x2 = rnorm(1000)
x3 = 0.6*x1 + 0.4*x2
x_matrix = cbind(x1, x2, x3)
x_cov = t(x_matrix)%*%x_matrix
delta = 0.6 * x_cov[,1] + 0.4 * x_cov[,2] - x_cov[,3]
```

注意, 在 R 中矩阵乘法使用 `%*`, 而 `x_matrix` 的转置矩阵是使用函数 `t` 得到的。这里我们使用 `set.seed` 函数来确定随机数生成器的种子以便于重复实验结果。在我们的模拟中可得

```
> delta
      x1      x2      x3
-5.684342e-13 -1.705303e-13  5.684342e-13
```

可以看出, $X^T X$ 的列是线性相关的 (这里 `delta` 在数值上已经非常小了), 因此这是一个奇异矩阵。由于矩阵 $X^T X$ 是不可逆的, 导致我们不能直接使用式 (5-9)。在实际中, $X^T X$ 可能是非奇异矩阵, 但其实非常接近奇异矩阵。在实际计算中, 如果一个矩阵越接近奇异矩阵, 那么计算它的逆矩阵的误差就越大^①。

为了保证计算的精确度, 同时降低模型的复杂度, 很多时候人们使用岭回归 (ridge regression)。在岭回归中, 我们计算 $X^T X + \lambda I$ 的逆矩阵, 这里 $\lambda > 0$ 是控制参数, I 是单位矩阵。简而言之, 我们就在矩阵 $X^T X$ 的所有对角元素上都加上了 λ , 从而使得计算逆矩阵 $(X^T X + \lambda I)^{-1}$ 的精度更高。

以上是从计算的角度来解释岭回归。我们还可以从损失函数和贝叶斯统计的角度来解释

① 事实上, 计算矩阵 A 的逆矩阵的精确度是由 A 的条件数 (condition number) 决定的。感兴趣的读者可以查阅参考文献[24]中相关章节内容。

岭回归。从贝叶斯统计的角度对岭回归的解释可参见参考文献[4]^①。下面我们再从损失函数的角度来解释岭回归。

严格地讲，岭回归最小化如下的损失函数：

$$L_2 = \sum_{i=1}^n \left(y_i - \sum_{j=1}^d w_j x_{ij} \right)^2 + \lambda \sum_{j=1}^d w_j^2 \quad (5-12)$$

写成矩阵的形式是：

$$L_2 = \|y - Xw\|_2^2 + \lambda \|w\|_2^2 = w^T (X^T X + \lambda I) w - 2w^T X^T y + y^T y \quad (5-13)$$

这里 $\lambda > 0$ 称为复杂度参数 (complexity parameter)，控制着模型的复杂度。复杂度参数在机器学习中有广泛的应用。我们在本书中将会广泛使用复杂度参数。在岭回归中，它直接控制着 w_j 的大小。 λ 越大，越多的 w_j 接近于 0。图 5-2 反映了随着 λ 的变化，我们使用岭回归处理 Boston housing data 数据^②时 w 中各个分量 w_j 的变化。在图 5-2 中，横坐标是 $\log(\lambda + 1)$ ，纵坐标是 w_j 的大小，每条曲线代表一个 w_j 的值在各个 λ 值下的变化。在这组数据中， X 的维数 d 是 13，因此有 13 条曲线。从图 5-2 中可以看出，当 λ 值较大时， w_j 都趋近于 0，而当 λ 较小时， w_j 所受的影响较小。从式 (5-12) 中可以看出，当 λ 较大时，损失函数中的第二项 $\lambda \sum_{j=1}^d w_j^2$ 影响更大，使得 w_j 的值都趋向于 0；而当 λ 较小时，损失函数的第一项影响较大，则 w_j 的值受第二项的影响较小。

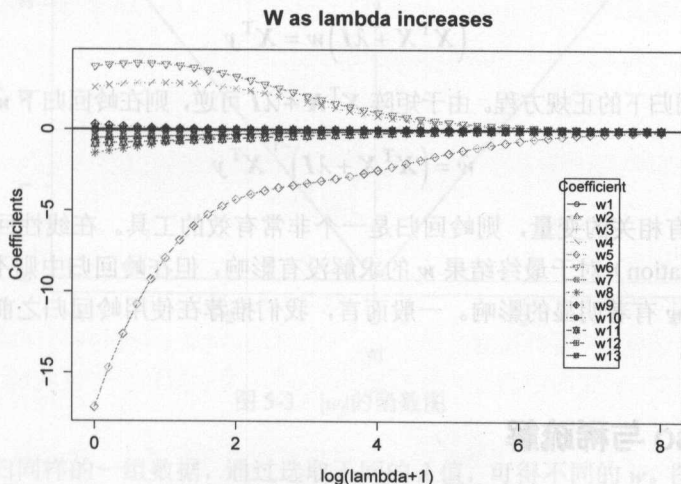


图 5-2 岭回归中 w_j 随着 λ 的变化情况

最小化式 (5-12) 中的损失函数等价于最小化如下条件约束问题：

① 在贝叶斯统计中，岭回归的基本假设是误差项 ε_i 的先验分布也是正态分布，本书不做详细推导。
② 数据可在 UCI 直接下载，网址为 <https://archive.ics.uci.edu/ml/datasets/Housing>。

$$\min_{\mathbf{w}} \sum_{i=1}^n \left(y_i - \sum_{j=1}^d w_j x_{ij} \right)^2 \quad (5-14)$$

$$\text{s.t. } \sum_{j=1}^d w_j^2 \leq \gamma$$

式(5-12)中的 λ 和式(5-14)中的 γ 存在一一对应关系。该等价关系可以由拉格朗日定理推得,在这里省略具体的证明过程。

类似地,通过最小化新的损失函数得到 \mathbf{w} 的最优解。由于式(5-12)中的函数仍然是关于 w_j 的二次函数,求得 L_2 关于 w_j 的导数并设为0即可找出使得 L_2 最小的 w_j 。这里我们给出 L_2 关于 w_j 的导数:

$$\frac{\partial L_2}{\partial w_j} = 2 \sum_{i=1}^n \left(y_i - \sum_{k=1}^d w_k x_{ik} \right) (-x_{ij}) + 2\lambda w_j \quad (5-15)$$

也可以写成矩阵的形式:

$$\frac{\partial L_2}{\partial \mathbf{w}} = 2\mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{X}^T \mathbf{y} + 2\lambda \mathbf{w} \quad (5-16)$$

令导数 $\frac{\partial L_2}{\partial \mathbf{w}}$ 为0,可得方程:

$$(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} = \mathbf{X}^T \mathbf{y} \quad (5-17)$$

该方程是岭回归下的正规方程。由于矩阵 $\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}$ 可逆,则在岭回归下 \mathbf{w} 的最优估计 $\hat{\mathbf{w}}$ 为

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \quad (5-18)$$

如果数据中有相关的变量,则岭回归是一个非常有效的工具。在线性回归中,数据是否标准化(normalization)对于最终结果 \mathbf{w} 的求解没有影响,但在岭回归中则不然。数据是否标准化对于最终的 \mathbf{w} 有着明显的影响。一般而言,我们推荐在使用岭回归之前首先标准化原始数据。

5.3.2 Lasso 与稀疏解

使用回归算法分析实际数据时,很多时候很多变量对于因变量 y 的预测其实是没有用处的,或者很多数据是冗余的。在这里,一个重要问题是如何从众多的变量中选取最有效的子集。在本节,我们讨论 Lasso^①。Lasso 与岭回归很相似,它们都在损失函数中添加正规化

① Lasso 的全称是 Least Absolute Shrinkage and Selection Operator, 参见 R. Tibshirani 所著《Regression Shrinkage and Selection via the Lasso》(Journal of the Royal Statistical Society, Series B, vol. 58, no. 1, pp. 267-288, 1996)。

项 (regularization)。在岭回归中, 我们使用的是 $\|\mathbf{w}\|_2^2 = \sum_{j=1}^d w_j^2$, 在 Lasso 中, 我们使用的则是 $\|\mathbf{w}\|_1 = \sum_{j=1}^d |w_j|$ 。严格地讲, 在 Lasso 中, 我们最小化如下的损失函数:

$$L_1 = \sum_{i=1}^n \left(y_i - \sum_{j=1}^d w_j x_{ij} \right)^2 + \lambda \sum_{j=1}^d |w_j| \quad (5-19)$$

这里 $\lambda > 0$ 是对应的复杂度参数。可以将 L_1 写成矩阵的形式:

$$L_1 = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 \quad (5-20)$$

事实上, L_1 也是一个凸函数 (因为 $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$ 和 $\|\mathbf{w}\|_1$ 都是关于 \mathbf{w} 的凸函数), 因此, L_1 的局部最优解也是它的全局最优解。但是这里最小化 L_1 的难点在于 L_1 在很多点是不可导的, 这样就不能使用前面求导数并使之等于 0 的方法来找出最小化 L_1 的最优解了。这里简单解释一下为什么 L_1 在有些点是不可导的。图 5-3 显示了 $|w_j|$ 的函数图, 可以看出 $|w_j|$ 在 $w_j = 0$ 处是连续的, 但不可导。那么对于 L_1 , 它在 d 维空间中任一 $w_j = 0 (j=1, 2, \dots, d)$ 的点都是不可导的。关于 Lasso 及相关问题的求解是前几年学术界研究的热点之一。这里不介绍如何求解 L_1 , 而直接介绍 R 中的 glmnet 包以求解 Lasso 问题。

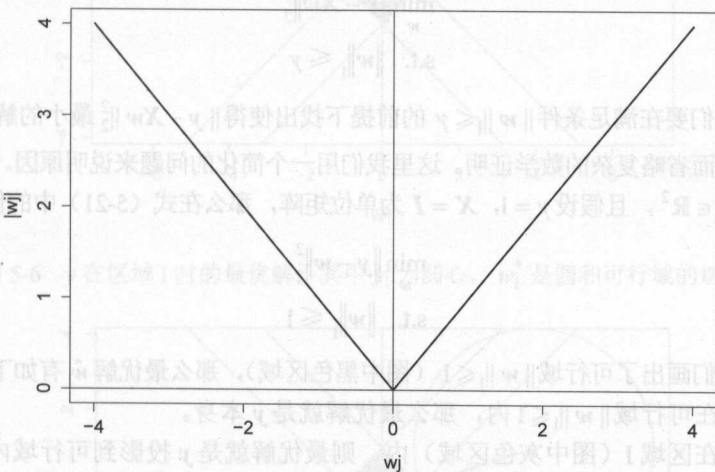
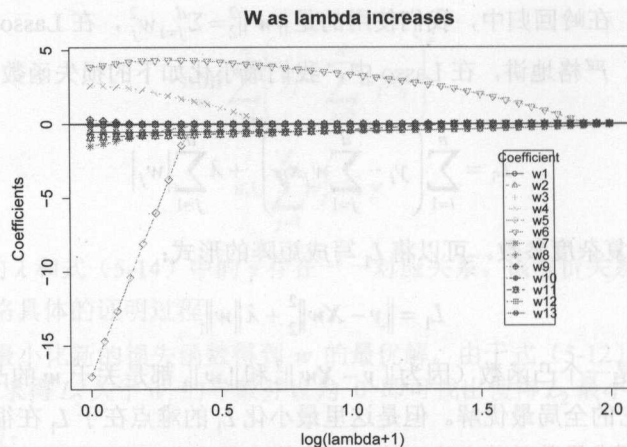


图 5-3 $|w_j|$ 的函数图

使用与岭回归同样的一组数据, 通过选取不同的 λ 值, 可得不同的 \mathbf{w} 。图 5-4 显示了不同的 λ 值对应的 \mathbf{w} 值。其中横坐标是 $\log(\lambda + 1)$ 值, 纵坐标是 w_j 的值, 每条曲线代表了 \mathbf{w} 的一个分量。从图 5-4 中可以看出, 当 λ 增大时, 越来越多的 w_j 变成了 0, 这就是通常所说的稀疏解 (sparse solution)。

图 5-4 Lasso 中 w_j 随着 λ 值的变化情况

为什么使用 L_1 范数 $\|\mathbf{w}\|_1$ 可以得到稀疏解呢？我们使用一个二维的例子来说明为什么 L_1 范数导致稀疏解。在这里需要了解一些关于优化算法的知识。根据拉格朗日定理，最小化损失函数 L_1 等价于优化如下优化问题：

$$\begin{aligned} \min_{\mathbf{w}} \quad & \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \\ \text{s.t.} \quad & \|\mathbf{w}\|_1 \leq \gamma \end{aligned} \quad (5-21)$$

换言之，我们要在满足条件 $\|\mathbf{w}\|_1 \leq \gamma$ 的前提下找出使得 $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$ 最小的解。我们试图给出比较直观的解释而省略复杂的数学证明。这里我们用一个简化的问题来说明原因。假设我们考虑二维的问题，即 $\mathbf{w} \in \mathbb{R}^2$ ，且假设 $\gamma = 1$ ， $\mathbf{X} = \mathbf{I}$ 为单位矩阵，那么在式 (5-21) 中的优化问题简化为：

$$\begin{aligned} \min_{\mathbf{w}} \quad & \|\mathbf{y} - \mathbf{w}\|_2^2 \\ \text{s.t.} \quad & \|\mathbf{w}\|_1 \leq 1 \end{aligned} \quad (5-22)$$

在图 5-5 中，我们画出了可行域 $\|\mathbf{w}\|_1 \leq 1$ （图中黑色区域），那么最优解 $\hat{\mathbf{w}}$ 有如下几种可能。

- 如果 \mathbf{y} 在可行域 $\|\mathbf{w}\|_1 \leq 1$ 内，那么最优解就是 \mathbf{y} 本身。
- 如果 \mathbf{y} 在区域 I（图中灰色区域）内，则最优解就是 \mathbf{y} 投影到可行域内的点。例如，在图 5-6 中， \mathbf{y}_1 （图中圆心）的投影就是 \mathbf{w}_1^* （图中圆和可行域的切点）。注意， \mathbf{w}_1^* 不是一个稀疏解。
- 如果 \mathbf{y} 在区域 II（白色区域）内，则最优解就是可行域的一个顶点。例如，在图 5-7 中， \mathbf{y}_2 （图中圆心）的投影就是 \mathbf{w}_2^* （图中圆和可行域的切点，也是可行域的一个顶点）。注意到 $\mathbf{w}_2^* = [1, 0]^T$ ，对应于一个稀疏解。

从此例可以看出，当 \mathbf{y} 在可行域内或者区域 I 内时，所得的解不是稀疏解；当 \mathbf{y} 在区域 II 内时，对应的解是稀疏解。当维度 d 增加时，区域 II 所对应的空间所占比例越来越高，从而导致得到稀疏解的可能性越来越大。

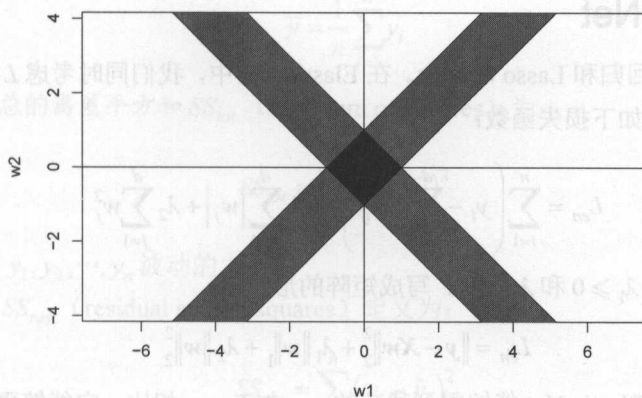


图 5-5 图中黑色区域是可行域，灰色区域是区域 I，而其他的白色区域为区域 II

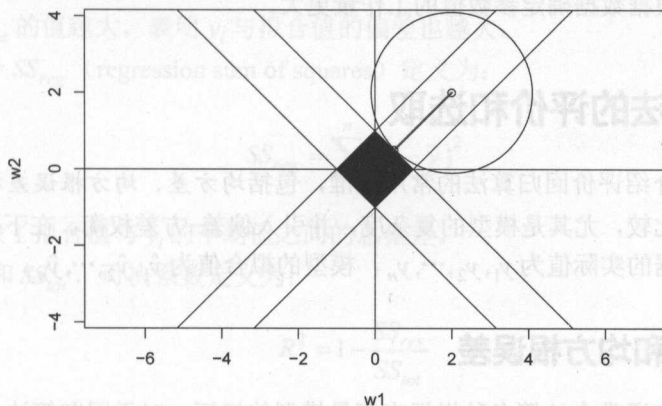


图 5-6 y 在区域 I 时的最优解。其中 y_1 为圆心， w_1^* 是圆和可行域的切点

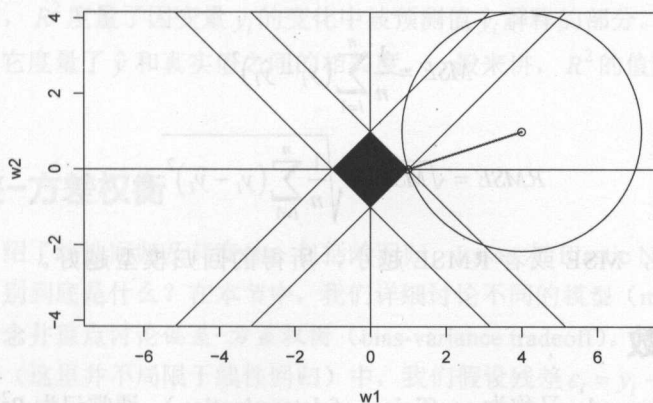


图 5-7 y 在区域 II 时的最优解。其中 y_2 为圆心， w_2^* 是圆和可行域的切点

5.3.3 Elastic Net

Elastic Net 是岭回归和 Lasso 的结合。在 Elastic Net 中, 我们同时考虑 L_1 范数和 L_2 范数。换言之, 我们最小化如下损失函数:

$$L_{en} = \sum_{i=1}^n \left(y_i - \sum_{j=1}^d w_j x_{ij} \right)^2 + \lambda_1 \sum_{j=1}^d |w_j| + \lambda_2 \sum_{j=1}^d w_j^2 \quad (5-23)$$

这里有两个复杂参数 $\lambda_1 \geq 0$ 和 $\lambda_2 \geq 0$ 。写成矩阵的形式是:

$$L_{en} = \|y - Xw\|_2^2 + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2 \quad (5-24)$$

与岭回归相比, Elastic Net 能够得到稀疏的 w ; 与 Lasso 相比, 它能够更加有效地处理成组的高相关的变量, 一般来说, 能够取得更高的预测性能。但是代价是我们引入了更多的控制参数, 在实际中根据数据确定参数值的工作量更大。

5.4 回归算法的评价和选取

本节首先简要介绍评价回归算法的常用标准, 包括均方差、均方根误差和可决系数。之后讨论不同模型的比较, 尤其是模型的复杂度, 并引入偏差-方差权衡。在下面的讨论中, 假设对于测试集中数据的实际值为 y_1, y_2, \dots, y_n , 模型的拟合值为 $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n$ 。

5.4.1 均方差和均方根误差

在模型评价中, 通常会计算多种指标来衡量模型的好坏。对于回归算法, 最常用的标准是均方差 (mean squared error, MSE) 和均方根误差 (root mean squared error, RMSE), 定义如下:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (5-25)$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (5-26)$$

在一般实践中, MSE 或者 RMSE 越小, 所得的回归模型越好。

5.4.2 可决系数

可决系数 (R-Squared, 又称为 coefficient of determination), 通常记为 R^2 。首先我们定义 \bar{y} 为:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i \quad (5-27)$$

那么定义数据的总的离差平方和 SS_{tot} (total sum of squares) 为:

$$SS_{tot} = \sum_{i=1}^n (y_i - \bar{y})^2 \quad (5-28)$$

SS_{tot} 反映了数据 y_1, y_2, \dots, y_n 波动的大小。

残差平方和 SS_{res} (residual sum of squares) 定义为:

$$SS_{res} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (5-29)$$

SS_{res} 反映了 y_i 与拟合值 \hat{y}_i 之间的差异。若 $SS_{res} = 0$, 表明每个 y_i 都可以由自变量的线性关系精确拟合; SS_{res} 的值越大, 表明 y_i 与拟合值的偏差也越大。

回归平方和 SS_{reg} (regression sum of squares) 定义为:

$$SS_{reg} = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 \quad (5-30)$$

回归平方和反映了拟合值与 y_i 的平均值之间的总偏差。

使用 SS_{res} 和 SS_{tot} , 可决系数定义为:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \quad (5-31)$$

根据可决系数的定义, 我们有 $R^2 \leq 1$ 。一般来讲, R^2 介于 0 和 1 之间。如果 $\hat{y}_i = \bar{y}$ 对于所有 i 都成立, 则 $R^2 = 0$; 如果 $\hat{y}_i = y_i$ 对于所有 i 都成立, 则 $R^2 = 1$ 。

从直观上讲, R^2 度量了因变量 y_i 的变化中被预测值 \hat{y}_i 解释的部分。事实上, R^2 是关于相关度的度量: 它度量了 \hat{y} 和真实值之间的相关度。一般来讲, R^2 的值越高, 所得的回归模型越好。

5.4.3 偏差-方差权衡

前面我们介绍了线性回归及其变体, 包括岭回归、Lasso 和 Elastic Net。那么不同的算法之间更深层的区别到底是什么? 在本节中, 我们详细讨论不同的模型 (model) 的区别, 引入模型复杂度的概念并重点讨论偏差-方差权衡 (bias-variance tradeoff)。

在回归分析 (这里并不局限于线性回归) 中, 我们假设残差 $\varepsilon_i = y_i - \hat{y}_i$ 的分布相互独立, 且均值为 0, 方差 σ^2 恒定。这里 ε_i 的分布可以为正态分布, 也可以是其他分布。那么, 模型的均方差 MSE 的期望值可以分解为:

$$E(MSE) = \sigma^2 + (\text{Model Bias})^2 + (\text{Model Variance}) \quad (5-32)$$

在式(5-32)中, $E(MSE)$ 是我们前面讨论的均方差的期望值, 它反映了模型的优劣。在式(5-32)中, 第一项 σ^2 通常认为是不可去除的错误项, 是数据中 y_i 的方差。这一项可以认为是数据本身的问题, 无论什么样的回归模型皆无法克服。第二项是模型偏差的平方, 它反映了该模型的函数形式能够以多精确的形式接近真正的数据。例如, 如果数据本身是由 $y = x^2$ 产生的, 用 x 来构建线性模型 $f(x) = ax + b$ 显然会导致偏差较大。最后一项是所谓的模型的方差。对于一组数据, 可以使用线性回归来拟合, 也可以使用二次函数甚至高次函数来拟合。显然, 二次函数的复杂度高于线性回归模型: 它需要更多的参数, 对于数据分布的假设也更多和更强, 但模型的表达能力更强。例如, 在这个例子中, 二次函数的表达能力显然比线性函数强: 线性函数是二次函数的一个特例。一般来讲, 表达能力越强的模型, 其方差也就越大。

图 5-8 中给出了一个实际例子来进一步阐述模型的偏差和方差。图 5-8 中画出了原始的数据集(由函数 $y = \cos x + \varepsilon$ 产生), 以及高偏差的模型和高方差的模型。在图 5-8 中, 黑色实线代表的是实际数据, 红色长虚线代表模型 1, 蓝色短虚线代表模型 2。模型 1 比较简单, 就是直接将 x 的取值区间分为 3 段, 每段用一条水平直线(对应这段数据的平均值)来拟合数据。该模型方差较低, 但是偏差较大。为什么我们说该模型方差较低呢? 例如, 又有一组也由 $y = \cos x + \varepsilon$ 产生的新的训练数据, 我们也用这种多段直线的模型去拟合, 模型本身的变化并不大。但是由图 5-8 也可以看出, 该模型过于简单, 对数据的拟合效果并不好, 偏差较高。而在蓝色的模型中, 我们用相邻前 3 点的平均来拟合, 即 $y_{i+3} = (y_i + y_{i+1} + y_{i+2})/3$ 。该模型对于这组数据拟合良好。但是, 如果新产生另一组数据, 模型本身就会发生较大变化。换言之, 就是模型偏差小, 但是方差大。

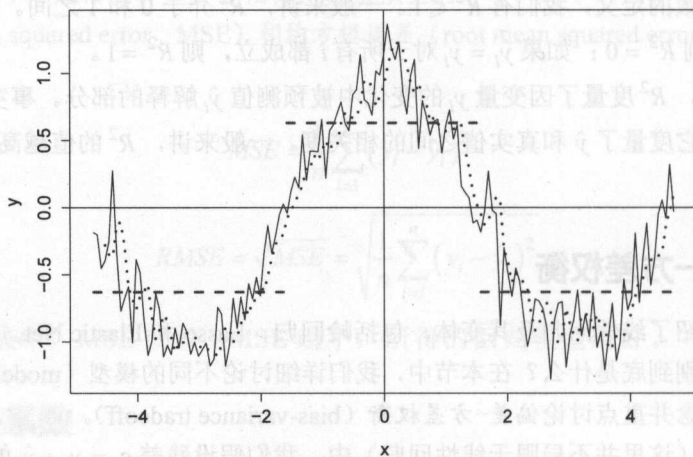


图 5-8 模型复杂度的实际例子

在我们讨论过的模型中,线性模型对应的 $(Model\ Bias)^2=0$,但是模型的方差较高。而岭回归、Lasso 和 Elastic Net 对应的 $(Model\ Bias)^2$ 不为 0,但是却将模型的方差缩小了,从而在整体上起到了降低式(5-32)中第二项和第三项之和的目的。注意,在岭回归、Lasso 和 Elastic Net 这些模型中,损失函数中添加的正则化项使得所得系数有向 0 靠近的趋势,从而限制了模型的表达能力。实质上是降低了模型的复杂度,减少了模型的方差。这种通过调节模型偏差和方差的方法称为偏差-方差权衡。

注意,并不是所有的数据都适合使用线性回归及其变体。在线性回归分析中,一个基本假设是因变量 y 和自变量 x 之间存在线性关系。以图 5-9 中的数据为例,图 5-9 (b) 中的数据显然不适合使用回归分析,但是图 5-9 (a) 中的数据是一个较好的回归分析的例子。

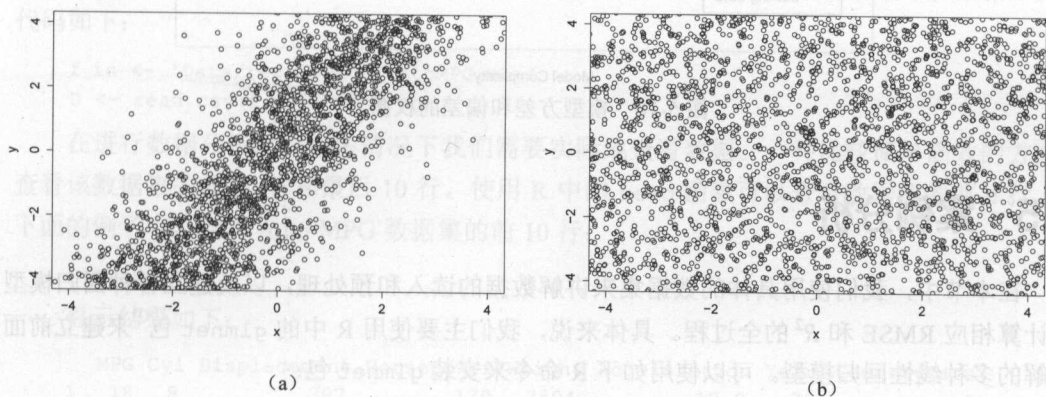


图 5-9 适用于及不适用于线性回归的数据

在本章中,我们主要讨论回归算法。事实上,在使用各类机器学习算法来分析数据时,一个普遍而且核心的问题就是:你需要多复杂的模型?或者换言之,如何选取合理的模型?一般而言,模型越复杂,方差越高,更容易在训练集上过拟合(over-fitting)。反过来,简单的模型不易于过拟合,但是由于模型表达能力有限,会导致欠拟合(under-fitting)的问题。在岭回归中,实际上牺牲了偏差来降低模型的方差。

图 5-10 显示了随着模型复杂度的增加,模型在训练集和测试集上的错误的变化。一般而言,我们在训练集上训练模型,然后计算所得模型在测试集上的性能。从图 5-10 中可以看出,当模型复杂度较低时,它在训练集和测试集上的错误都较大。此时模型的偏差较大,但是模型的方差较小。当模型的复杂度升高时,它在训练集上的错误会持续下降。但是,模型在测试数据上的错误一般会先下降;当模型在训练集上过拟合之后,我们可以看到模型在测试数据上的错误不降反升。在这一过程中,随着模型复杂度的升高,模型的偏差越来越小,但是模型的方差却越来越大。从图 5-10 中也可以看出,为了在测试集上取得最优的结果,模型的复杂度并不是越大越好。

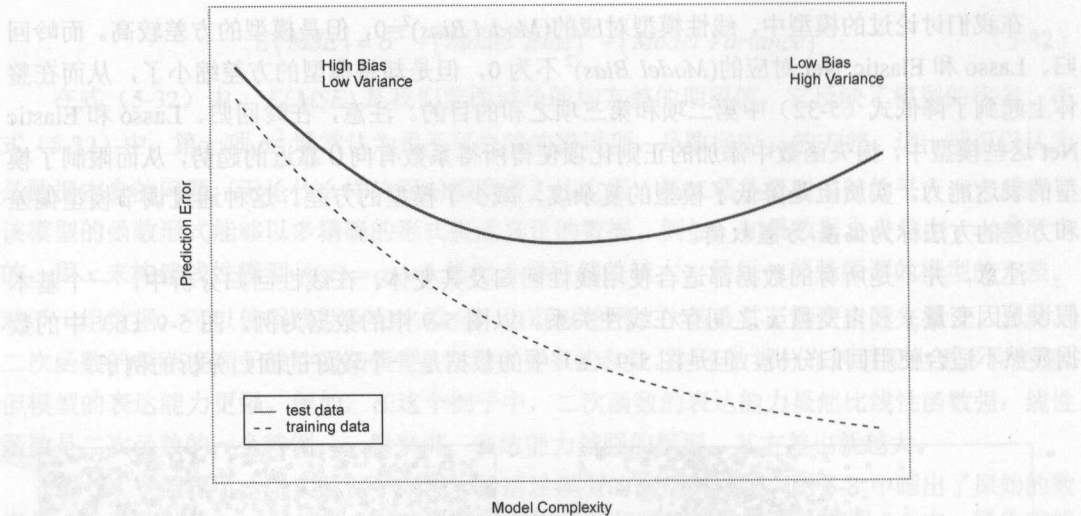


图 5-10 模型方差和偏差的权衡

5.5 案例分析

在本节中，我们使用具体的数据集来讲解数据的读入和预处理，以及建立各种回归模型并计算相应 RMSE 和 R^2 的全过程。具体来说，我们主要使用 R 中的 glmnet 包^①来建立前面讲解的多种线性回归模型。可以使用如下 R 命令来安装 glmnet 包：

```
install.packages('glmnet', dependencies=T)
```

关于包的安装和使用更多的细节可参见第 2 章。本节的所有代码都在文件 Auto_MPG_regression.R 中。感兴趣的读者可以直接下载并运行该程序。

5.5.1 数据导入和探索

在本例中，我们使用 Auto-MPG 数据^②。在该数据中，使用 8 个特征来预测汽车的 MPG (miles per gallon)，即每加仑^③汽油可行驶英里数。表 5-1 总结了 Auto-MPG 数据集的各原始变量、含义及其类型。

表 5-1 Auto-MPG 数据集的各原始变量、含义及其类型

特征名	含义	类型
MPG	每加仑可行驶英里数	数值变量
Cyl	发动机气缸数	数值变量

① <https://cran.r-project.org/web/packages/glmnet/index.html>

② 该数据集可在 <https://archive.ics.uci.edu/ml/datasets/Auto+MPG> 下载。

③ 加仑是容量单位，1 加仑=3.78 升。

续表

特 征 名	含 义	类 型
Displacement	发动机排量	数值变量
Horsepower	发动机马力	数值变量
Weight	车辆重量	数值变量
Acceleration	0~60 英里/小时加速时间	数值变量
Year	车辆年份	数值变量
CountryCode	车辆原产地代码	分类变量
Model	车辆型号	字符变量

由于数据是以 csv 文件提供的, 因此可以使用 R 中的 `read.csv` 函数来导入数据。具体代码如下:

```
f_in <- 'Data/regression_autoMPG.csv'
D <- read.csv(f_in)
```

在进行数据分析时, 很多情况下我们需要实际地查看数据。一种比较简单高效的方式是查看该数据集的前 10 行和最后 10 行。使用 R 中的 `head` 函数可以查看数据集的前 n 行。在下面的例子中, 查看 Auto-MPG 数据集的前 10 行:

```
head(D, 10)
```

显示结果如下:

```

      MPG Cyl Displacement Horsepower  Weight  Acceleration  Year  CountryCode
1    18   8         307         130   3504         12.0      70         1
2    15   8         350         165   3693         11.5      70         1
3    18   8         318         150   3436         11.0      70         1
4    16   8         304         150   3433         12.0      70         1
5    17   8         302         140   3449         10.5      70         1
6    15   8         429         198   4341         10.0      70         1
7    14   8         454         220   4354          9.0      70         1
8    14   8         440         215   4312          8.5      70         1
9    14   8         455         225   4425         10.0      70         1
10   15   8         390         190   3850          8.5      70         1
      Model
1 chevrolet chevelle malibu
2   buick skylark 320
3 plymouth satellite
4   amc rebel sst
5   ford torino
6   ford galaxie 500
7 chevrolet impala
8 plymouth fury iii
9 pontiac catalina
10 amc ambassador dpl
```

类似地, 可以使用 `tail` 函数查看该数据集的最后 10 行:

```
tail(D, 10)
```


还可以利用 `class` 函数检查读入的数据框 `D` 中每列的数据类型:

```
for(i in 1:ncol(D)) {
  msg <- paste('col ', i, ' and its type is ', class(D[,i]))
  print(msg)
}
```

对应的输出如下:

```
[1] "col 1 and its type is numeric"
[1] "col 2 and its type is integer"
[1] "col 3 and its type is numeric"
[1] "col 4 and its type is integer"
[1] "col 5 and its type is integer"
[1] "col 6 and its type is numeric"
[1] "col 7 and its type is integer"
[1] "col 8 and its type is integer"
[1] "col 9 and its type is factor"
```

此外, 还可以使用 `str` 函数来简洁地打印数据框 `D` 的相关信息。在这个例子中, `str(D)` 的输出如下:

```
> str(D)
'data.frame' : 392 obs. of 9 variables:
 $ MPG      : num  18 15 18 16 17 15 14 14 15 ...
 $ Cyl      : int   8 8 8 8 8 8 8 8 8 ...
 $ Displacement: num  307 350 318 304 302 429 454 440 455 390 ...
 $ Horsepower : int   130 165 150 150 140 198 220 215 225 190 ...
 $ Weight     : int  3504 3693 3436 3433 3449 4341 4354 4312 4425 3850 ...
 $ Acceleration: num   12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
 $ Year      : int   70 70 70 70 70 70 70 70 70 70 ...
 $ CountryCode : int    1 1 1 1 1 1 1 1 1 1 ...
 $ Model     : Factor w/ 301 levels "amc ambassador brougham",...: 49 36 230
14 160 141 54 222 240 2 ...
```

5.5.2 数据预处理

当使用 `read.csv` 函数读入 Auto-MPG 数据集时, 除了最后一列 `Model` 因为含有字符被处理为因子类型外, 其他列都被转化为数值类型。变量 `CountryCode` 由于其值属于集合 $\{1,2,3\}$, 因此也被 R 默认为数值类型。事实上, `CountryCode` 是分类变量, 因为它的值只能是 1、2、3 之一, 其中:

- 1 表示美国;
- 2 表示欧洲;
- 3 表示日本。

这里的 1~3 之间并没有大小的关系, 只是一个代码而已。注意到在使用线性回归时, 我们要求所有的变量都是数值型的。因此, 还需要进一步将 `CountryCode` 变量转换为数值变量才能使用本章讨论的回归算法。后续章节将介绍可以直接处理分类变量的算法, 如决策树、随机森林等。

将分类变量转换为数值变量的一个常用方法是将其转换为哑变量。我们在第 4 章中已经

讨论过哑变量。通常情况下, 如果一个分类变量有 k 个不同的值, 则引入 $k-1$ 个哑变量表示不同的取值。注意, 如果引入 k 个哑变量, 则会引起变量的共线性, 在线性回归中会导致所得的矩阵 $\mathbf{X}^T \mathbf{X}$ 不可逆。在此例中, 我们将 CountryCode 转换为 2 个哑变量 CountryCode1 和 CountryCode2, 其中 CountryCode1 为 1 表示原来的 CountryCode 为 1, CountryCode2 为 1 表示原来的 CountryCode 为 2。具体的转换关系可参见表 5-2。

表 5-2 CountryCode 变量转化为哑变量

CountryCode	CountryCode1	CountryCode2
1	1	0
2	0	1
3	0	0

在具体的实现中, 使用如下 R 代码完成下面的转换。首先生成 CountryCode1 和 CountryCode2 变量, 然后删除原来的 CountryCode 变量。在我们提供的 R 代码中, utility_funcs.R 文件中的函数 categorical2binary 能够自动将分类变量转化为哑变量。同时在 Auto-MPG 数据中, 因为 Model 变量的用处不大, 所以也直接删除该列。

```
D$CountryCode1 <- ifelse(D$CountryCode==1, 1, 0)
D$CountryCode2 <- ifelse(D$CountryCode==2, 1, 0)
D$CountryCode <- NULL
D$Model <- NULL
```

5.5.3 将数据集分成训练集和测试集

在机器学习中, 通常将数据集分为训练集和测试集: 通过训练集学习所要的模型; 通过测试集可以测试所得的模型的性能。在下面的例子中, 我们随机选取 70% 的数据作为训练集, 并将剩余的 30% 作为测试集。

```
train_ratio <- 0.7
n_total <- nrow(D)
n_train <- round(train_ratio * n_total)
list_sample <- sample(nrow(D))
list_train <- list_sample[1:n_train]
list_test <- list_sample[(n_train+1):n_total]
D_train <- D[list_train,]
D_test <- D[list_test,]
y_train <- D_train$MPG
y_test <- D_test$MPG
```

有时候我们能够重复实验结果, 在调用 sample 函数之前可以使用 set.seed 函数来设定随机数生成器的种子。注意, 训练集和测试集的划分是随机的, 读者在运行下面的程序时所得结果可能稍有不同。

5.5.4 建立一个简单的线性回归模型

R 中提供的 lm 函数可以用来构造一个简单的线性回归模型。生成所得模型后, 可以使用

`predict` 函数直接得到测试数据上的预测。我们在文件 `regression_measures.R` 中提供的函数 `R2_score` 和 `RMSE` 可以分别计算所得预测在测试集上的 R^2 和 `RMSE`。

```
M1 <- lm(MPG~., data = D_train)
y_predict1 <- predict(M1, newdata=D_test)
source('regression_measures.R')
R2_m1 <- R2_score(y_test, y_predict1)
rmse_m1 <- RMSE(y_test, y_predict1)
```

在使用 `lm` 函数时, 我们使用公式 `MPG~.`, 表示指定 `MPG` 是我们要预测的目标变量, 并且使用 `D_train` 中的所有其他变量作为自变量。函数 `lm` 返回已经训练好的模型 `M1`, 这样就可以直接使用 `predict` 函数来预测测试集 `D_test` 上的 `MPG`。在使用 `predict` 函数时, 需要将 `newdata` 参数设置为 `D_test`。

在 R 中, 可以绘制出散点图来比较测试集上的预测值和真实值。图 5-11 显示了所得的散点图。在图 5-11 中还增加了一条直线 `y_predict1=y_test` 来观察真实值和预测值之间的差距。下面是用来画图的 R 代码:

```
plot(y_test, y_predict1, type='p')
abline(c(0,1), col='red')
```

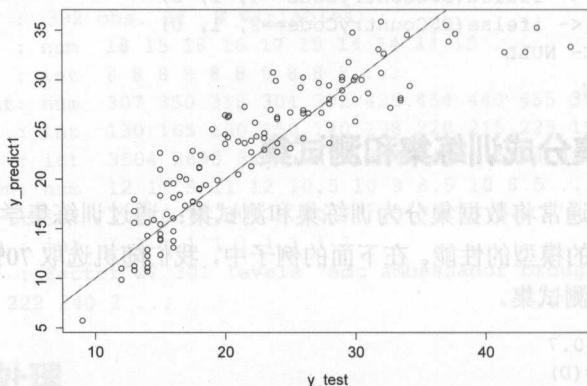


图 5-11 测试集上真实值和预测值的比较 (其中 x 轴对应真实值, y 轴对应 `lm` 模型的预测值。同时直线 $y=x$ 也在图中标出)

5.5.5 建立岭回归和 Lasso 模型

下面将介绍更加强大的 `glmnet` 包。该包涵盖了常用的适用于分类和回归的线性模型, 包括:

- 线性回归及变体, 包括岭回归、Lasso 和 Elastic Net;
- 逻辑回归及其变体 (将在第 6 章讨论)。

下面给出使用 `glmnet` 进行岭回归、Lasso 和 Elastic Net 的具体例子。`glmnet` 包同时支持 L_1 范数和 L_2 范数的优化。换言之, 对本章介绍的回归算法, 我们都可以使用 `glmnet` 包来完成计算。互联网上有关于 `glmnet` 包的详细介绍^①, 本书将介绍 `glmnet` 包的常用方法。

① <http://cran.r-project.org/web/packages/glmnet/glmnet.pdf>

在使用 glmnet 包进行回归分析时, 主要使用如下两个函数:

- glmnet() 函数, 该函数名与包的名字相同, 主要用于利用训练数据建立模型;
- predict() 函数 (实质上是 predict.glmnet 函数), 利用该函数和已经得到的模型, 可以对新数据进行预测。

在 glmnet 函数中, L_1 范数和 L_2 范数的权重由参数 alpha 和 lambda 来确定。其中, L_1 范数的权重为 $\lambda \times \alpha$, 而 L_2 范数的权重为 $\lambda \times (1 - \alpha) / 2$ 。注意, 在本章中, 我们将 glmnet 函数的 family 参数设为 'gaussian', 表示要使用的是线性回归算法。事实上 glmnet 也支持逻辑回归, 只需要将 family 参数设置为相应的值即可。我们在第 6 章会更加详细地讨论 glmnet 函数的用法。

在下面的例子中计算岭回归, 并将 L_2 范数的权重设为 0.25。注意, 在岭回归中, 需要将参数 alpha 设为 0。在使用 glmnet 时, 必须将自变量表示为矩阵的形式。由于 D_train 是数据框类型, 因此需要使用 as.matrix 函数将其显式地转换为矩阵的形式。glmnet 的输出是一个 glmnet 类, 我们所要的解最终保存在 w2_1 中。

```
family <- 'gaussian'
alpha <- 0
lambda <- 0.5
M2_1 <- glmnet(as.matrix(D_train[,2:ncol(D_train)]),
               y_train, family = family,
               lambda=lambda, alpha=alpha)
W2_1 <- M2_1$beta
```

下面这个例子计算 Lasso, 并将 L_1 范数的权重设为 1。由于只需要设置 L_1 范数的权重, 因此将 alpha 和 lambda 都设为 1。下面是对应的 R 代码:

```
family <- 'gaussian'
alpha <- 1
lambda <- 1
M1_1 <- glmnet(as.matrix(D_train[,2:ncol(D_train)]),
               y_train, family = family,
               lambda=lambda, alpha=alpha)
W1_1 <- M1_1$beta
```

所得的解 w1_1 为稀疏解, 表 5-3 是所得 w1_1 的值。从表 5-3 中可以看出, w1_1 有 4 个分量的值为 0。

表 5-3 w1_1 中各分量的值

分 量	值
Cyl	0
Displacement	0
Horsepower	-0.00309348156260112
Weight	-0.00543817447042681
Acceleration	0
Year	0.544922253902601
CountryCode1	-0.784951577206556
CountryCode2	0

类似地，可以同时使用 L_1 范数和 L_2 范数，从而得到 Elastic Net 模型。在下面的例子中， L_1 范数的权重为 1， L_2 范数的权重为 2。

```
family <- 'gaussian'
alpha <- 0.2
lambda <- 5
M_elastic <- glmnet(as.matrix(D_train[,2:ncol(D_train)]),
                    y_train, family = family,
                    lambda=lambda, alpha=alpha)
W_elastic <- M_elastic$beta
```

在所有的这些例子中，都可以使用 predict 函数来得到训练所得模型在测试集上的预测值。

```
y_predict_elastic <- predict(M_elastic,
                             newx=as.matrix(D_test[,2:ncol(D_test)]))
```

与 lm 函数稍稍不同，在这里的 predict 函数中我们需要使用 newx 参数来指定测试集的数据。注意，与 glmnet 函数类似，这里输入自变量的时候也需要显式地转换为矩阵的形式。

5.5.6 选取合适的模型

在实际使用回归算法时，大多数时候我们不知道 L_1 范数和 L_2 范数的具体权重。更多时候，我们需要测试一系列权值并从中选取最优的模型。glmnet 的好处是可以直接输入一系列的参数值，并自动计算出每个参数值对应的解。

在下面这个例子中，我们测试岭回归，并将 lambda 的值从 1 以 0.05 的步长递减到 0（共 21 个值）。glmnet 对于每个 lambda 值都建立一个线性模型并将其存于 M_ridge_list 中。最后我们使用 predict 函数，使用所得的 21 个模型对测试集进行处理。所得的 y_predict_test_ridge 是一个矩阵，每一列对应一个模型的预测值。通过同时测试多个参数，我们可以选取最优的 lambda 值。一般来说，在机器学习中，我们通常使用交叉检验（cross-validation）将训练集分为多个子集组成训练/测试集，并测试多个参数，最后选取性能最好的参数。在回归分析中，我们可以使用可决系数或者 RMSE 作为选取标准。我们将在第 6 章详细讨论交叉检验的原理和具体流程。在这里，我们直接比较各个 lambda 值所得的模型在测试集上的 RMSE，并保存在向量 RMSE_test_list_ridge 中。

```
lambda_list <- seq(1, 0, by=-0.05)
lambda_num <- length(lambda_list)
X_train <- as.matrix(D_train[, 2:ncol(D_train)])
X_test <- as.matrix(D_test[, 2:ncol(D_test)])
RMSE_test_list_ridge <- rep(0, lambda_num)
family <- 'gaussian'
alpha <- 0
M_ridge_list <- glmnet(X_train, y_train,
                      family = family,
                      lambda=lambda_list,
```

```

        alpha=alpha)
y_predict_test_ridge <- predict(M_ridge_list, X_test)
for (i in 1:lambda_num) {
  RMSE_test_list_ridge[i] <- RMSE(y_test,
                                y_predict_test_ridge[,i])
}

```

这里我们将计算 RMSE 的函数 RMSE 保存在文件 regression_measures.R, 因此, 在前面我们使用 source('regression_measures.R') 来载入该函数。

类似地, 可以使用同样的 lambda 值来训练 Lasso 模型。R 代码如下。主要区别在于将 alpha 的值设为 1。

```

RMSE_test_list_lasso <- rep(0, lambda_num)
family <- 'gaussian'
alpha <- 1
M_lasso_list <- glmnet(X_train, y_train,
                      family = family,
                      lambda=lambda_list,
                      alpha=alpha)
y_predict_test_lasso <- predict(M_lasso_list, X_test)
for (i in 1:lambda_num) {
  RMSE_test_list_lasso[i] <- RMSE(y_test,
                                y_predict_test_lasso[,i])
}

```

图 5-12 显示了岭回归和 Lasso 两种算法在测试集上 RMSE 的变化。注意, 当 lambda 的值为 0 时, 就是线性回归。从图 5-12 中可以看出, 当 lambda 为 0.35 时, 岭回归和 Lasso 同时取得最好的 RMSE, 但是 Lasso 的表现更好。从图 5-12 中还可以看出, 岭回归的 RMSE 对 lambda 不是很敏感, 但是 Lasso 则比较敏感。在实际中, 一定要根据自己的实际情况选择合适的模型。

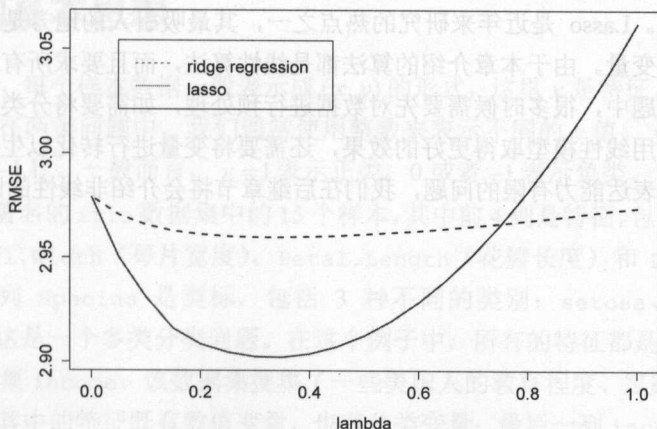


图 5-12 岭回归和 Lasso 在测试集上的 RMSE 比较

5.5.7 构造新的变量

在使用线性回归或者相关算法时，我们假设目标和变量之间存在线性关系。如果实际数据中线性关系并不存在的话，我们并不能直接使用线性回归模型。相应地，我们可以根据现有数据构造更多的变量，然后使用线性回归模型。一个简单的例子是假设我们有一系列由 $y = 4x^2 - 1$ 生成的数据集。显然，用关于 x 的线性模型来预测 y 不能解决问题。但是，如果我们用关于 x^2 的线性模型来预测 y ，则可以取得很好的性能表现。

在实际中，应根据数据的具体特征和具体的问题来构造新的变量。此外，一定要关注变量间的相互关系。例如，因变量 y 与变量 x_1 、 x_2 都不存在线性关系，但是与 x_1x_2 存在线性关系。在这种情况下，新变量 x_1x_2 对于提升线性回归性能的效果非常明显，但是需要读者对于数据有更加深入的了解和认识。

5.6 小结

本章介绍了几种基本的回归算法，同时也引入了机器学习中的许多重要概念，包括：

- 损失函数；
- 模型的偏差-方差权衡；
- 模型的复杂度；
- 模型的评价和选取；
- 变量的选取。

在后面的章节中，这些概念还会被反复提到。举个简单的例子，通过选用不同的损失函数，可推导得到一种常用的分类算法——逻辑回归。

对于本章所介绍的线性回归算法，在实际中使用最多的是岭回归。它不仅计算简单，而且能取得较好的效果。Lasso 是近年来研究的热点之一，其最吸引人的地方是能够得到稀疏解，从而选取最有用的变量。由于本章介绍的算法都是线性算法，而且要求所有的变量都是数值型的，因此在实际问题中，很多时候需要先对数据进行预处理，如需要将分类变量转化为哑变量等。此外，为了使用线性模型取得更好的效果，还需要将变量进行转化以生成更多的新变量。为了克服线性模型表达能力有限的问题，我们在后继章节将会介绍非线性的回归算法。

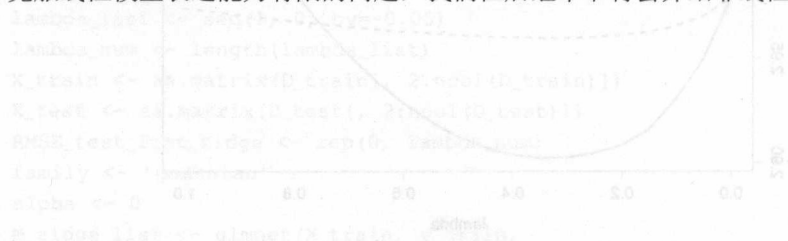


图 5-12 岭回归和 Lasso 回归的交叉验证 R-squared 值

第 6 章 分类算法

在分类 (classification) 中, 我们的任务是将对象 (object) 归类到已经定义好的若干类中。如果只有两类, 则称为两类分类 (binary classification)。如果多于两类, 则称为多类分类 (multi-class classification)。与在回归问题中的讨论类似, 每个对象都由一系列特征描述。

分类算法在实际中有大量的应用, 同时它也是很多更复杂算法的基础。本章介绍分类问题的基本思想及常用的分类算法, 特别是以下 3 种算法:

- 决策树;
- 逻辑回归;
- 支持向量机。

本章首先讨论决策树。注意, 决策树不是线性分类器, 但它原理简单, 同时也是很多复杂分类算法的基础, 因此首先予以讨论。此外, 我们还会介绍损失函数 (loss function), 并讨论不同的分类算法如何对应不同的损失函数, 以及如何使用正则化项来控制模型的复杂度。与回归算法的讨论类似, 我们也讨论分类算法的诸多评价标准。

为了解决实际中更复杂的分类问题, 我们将会详细讨论如何解决不平衡分类问题。在本章中, 我们同时会介绍 R 中对应的软件包, 这样读者可以直接尝试使用各种分类算法。特别地, 我们将介绍 caret 包, 这样读者能够简单地使用交叉检验来为机器学习中的大量常用算法选取最优参数。

6.1 分类的基本思想

在分类问题中, 每个样本数据一般表示成 (\mathbf{x}, y) 的形式, 这里 \mathbf{x} 是特征, 而 y 是对应的类标 (class label)。在两类问题中, 我们通常使用整数来表示不同的 y 值。 y 不同的取值包括 $y \in \{0, 1\}$ 或者 $y \in \{-1, 1\}$ 。一般而言, $y = 1$ 表示正类, 0 或者 -1 表示负类。

表 6-1 给出了著名的 iris 数据集中的 15 个样本, 其中前 4 列是特征, 包括 Sepal.Length (萼片长度)、Sepal.Width (萼片宽度)、Petal.Length (花瓣长度) 和 Petal.Width (花瓣宽度); 最后一列 Species 是类标, 包括 3 种不同的类别: setosa、versicolor 和 virginica, 因此这是一个多类分类问题。在这个例子中, 所有的特征都是数值变量。表 6-2 给出了另一个数据集 income。该数据集搜集了一些美国人的教育程度、家庭状况、工作时间和年收入等数据。其中的特征既有数值变量, 也有分类变量; 最后一列 income 是类标, 有两个不同的值 $\leq 50K$ 和 $> 50K$, 因此这是一个两类分类问题。

表 6-1 iris 数据集 (其中前 4 列是特征, 最后一列是类标)

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4.0	1.3	versicolor
6.5	2.8	4.6	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3.0	5.9	2.1	virginica
6.3	2.9	5.6	1.8	virginica
6.5	3.0	5.8	2.2	virginica

在分类中, 我们可以将第 i 个对象的特征用 d 维向量 $\mathbf{x}_i \in \mathbb{R}^d$ 表示。在分类问题中, 我们的目标是要根据每个样本的特征 \mathbf{x} 构建一个函数 $f(\mathbf{x})$, 使得 $f(\mathbf{x})$ 能够输出 \mathbf{x} 对应的类标。同回归问题的处理类似, 在实践中通常也将数据分为训练集 (training set) 和测试集 (test set)。利用训练集, 使用学习算法得到相应的模型。然后, 可以将模型应用到测试集上, 估计所得模型的性能。

根据上面的描述, 我们可以知道分类问题和回归问题是非常相似的。它们的区别在于, 在分类问题中, 类标必须是离散的, 而在回归问题中, 目标变量是连续的。

在解决分类问题时, 不同的算法有不同的策略。一类方法称为“懒学习”。在懒学习中, 我们在得到了训练集时什么也不干。等到测试集到来时, 我们再利用训练集对测试集进行处理。懒学习的典型例子是 k 近邻 (k -Nearest Neighbor, k NN) 算法。下面我们简单地讨论一下 k NN 算法。

在 k NN 中, 测试集中每个样本的类别由训练集中与它最相似的 k 个样本决定。在得到最相似的 k 个样本后, 我们使用投票的方法将 k 个样本中出现频率最高的类标作为该样本的类标。通常在 k NN 中我们需要保存整个训练集并且要有效地为新样本从训练集中找到最相似的样本, 一般来讲, 算法的效率较低。如果训练集中某些样本的类标不准确或者存在噪声的话, 都会影响最终的分类结果。

与懒学习相对的是“积极学习”。在积极学习中, 我们首先挖掘训练数据, 从中构建相应的模型。而在懒学习中, 我们拖延学习这一过程直至测试集中新的样本到来。在积极学习中, 当新的样本到来时, 我们直接使用训练所得的模型得到相应的结果。因此, 在积极学习中, 主要的时间花在了如何根据训练集训练模型。一旦得到模型, 处理新的样本一般非常迅速。

本章介绍的分类方法都是积极学习的方法。

表 6-2 income 数据集 (其中最后一列是类标, 表示该样本的年收入是否高于 5 万美元)

age	workclass	fnlwgt	education	ed-num	marital-status	occupation	relations-hip	race	sex	capita-l-gain	capita-l-loss	hours-per-week	native-country	income
39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	≤50K
50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	≤50K
38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	≤50K
53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	≤50K
28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	≤50K
37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	≤50K
49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica	≤50K
52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45	United-States	>50K
31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-family	White	Female	14084	0	50	United-States	>50K
42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	5178	0	40	United-States	>50K

6.2 决策树

本节我们讨论决策树 (decision tree)。决策树原理简单, 是处理分类问题的最基本的算法之一。在实际中, 我们很少直接使用决策树来处理分类问题。但多种基于决策树的分类算法, 如随机森林和提升树, 因其简单易用和良好的性能而在工业界得到了广泛应用。因此, 本节着重介绍决策树的基本原理, 为以后介绍随机森林和提升树打好基础。

6.2.1 基本原理

顾名思义, 决策树就是将规则以树的形式组织起来, 从而对样本进行分类。在决策树中, 有以下两种类型的结点。

- 非叶结点: 每个非叶结点对应于一个特征或者属性, 且有两个或多个子结点。
- 叶结点: 每个叶结点对应一个类别。

在决策树中, 每条边对应着一个判定条件。

图 6-1 给出了根据 iris 数据集, 使用 R 中的 rpart 包构建的决策树。在该决策树中, 根结点对应于特征 Petal.Length, 根结点所连接的两条边分别对应判定条件 $\text{Petal.Length} < 2.45$ 和 $\text{Petal.Length} \geq 2.45$ 。最左下角的叶结点所对应的类标是 setosa。

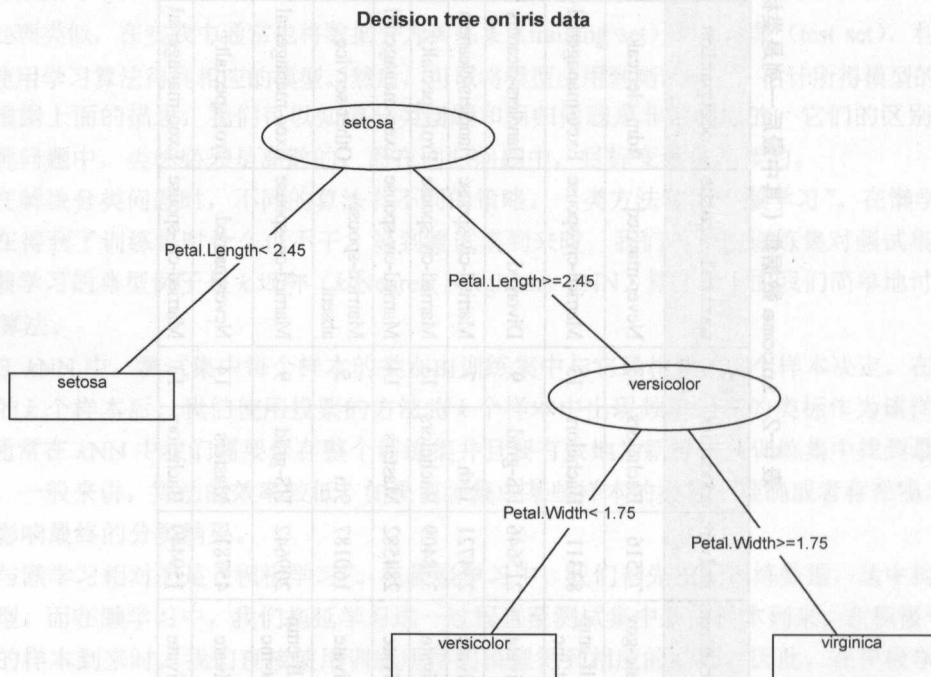


图 6-1 使用 R 中的 rpart 软件包对 iris 数据集生成的决策树

利用已有的决策树，可以很简单地将一个新的样本分类。基本原理是从根结点出发，不断地测试对应的特征和相关的条件，直到到达一个叶结点为止。这样，我们可以将叶结点所对应的类标作为该样本的类标。决策树的计算复杂度低，特别是对新数据进行判定时非常快。

下面我们简单讨论一下如何利用图 6-1 中的决策树将实际样本分类。考虑表 6-3 中的第一个测试样本。首先我们考虑根结点对应的特征 `Petal.Length` 和条件 `Petal.Length < 2.45`。由于该样本对应的 `Petal.Length` 值为 1.4，因此我们要沿着根结点左侧的边到达相应结点。注意，该结点是叶结点且对应的类标为 `setosa`，按照该决策树，我们应该将该样本分类为 `setosa`。类似地，我们可以应用该决策树处理表 6-3 中的第二个样本。由于该样本对应的 `Petal.Length` 值为 6.0，因此，从根结点出发，我们应该沿着右侧的边到达相应结点。该结点不是叶结点，其对应的特征是 `Petal.Width`。由于该样本对应的 `Petal.Width` 值为 2.5，满足右侧的边所对应的条件（即 `Petal.Width >= 1.75`），因此我们应该沿着右侧的边到达最右下角的叶结点。注意，该结点对应的类标是 `virginica`，因此，我们应该将该样本分类为 `virginica`。可以看出，简单的决策树模型成功地分类了表 6-3 中的两个样本。

表 6-3 测试样本

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
6.3	3.3	6.0	2.5	virginica

6.2.2 决策树学习

本节讨论如何从训练数据中学习得到决策树模型。

从理论上讲，对于一个给定的训练数据集，由于可能的决策树模型太多，因此在实际中我们无法遍历所有可能的决策树以确定最优的决策树。这里我们可做一个简单的估算。假设输入数据有 d 个特征（或属性），则第一层的根结点有 d 种不同的选择。在决定了根结点对应的特征后，假设后面我们不选择重复的属性。第二层中的每个非叶结点对应的属性又有 $d-1$ 种不同的选择。因此，所有可能的决策树的总数是 d 的指数级别的。

实际中，为了有效地构建决策树模型，我们通常采用自顶向下的“贪心”算法。在这里我们给出贪心算法的基本流程。注意，在不同的决策树算法（如 C4.5 和 CART）中，生成决策树的算法稍有不同，这里我们仅介绍一般性的流程。

在使用训练集训练决策树时，每个结点都对应着训练集的一个子集。首先，我们将整个训练集都赋给根结点；然后，考虑所有的属性及判定条件，使得训练集能够被划分为两个或多个部分，并为每部分构造一个新的结点；最后，递归地处理每个新结点，直到不能再划分为止。在每次划分后，我们希望每部分的训练集“纯度”尽可能高一些。最理想的情况是：如果问题是两类分类问题，划分后我们希望每部分都只包含一类的所有样本。具体的步骤如算法 6-1 所示。

算法 6-1 决策树的生成

- (1) 创建空的结点集合 N 、边集合 E 。
- (2) 创建根结点, 将其加入集合 N , 并将整个训练集赋给该结点。
- (3) 对集合 N 中的每个未处理结点 i , 设其所对应的训练集为 D_i :
 - 如果 D_i 中的样本都属于同一类, 则将结点 i 标识为叶结点, 且将该结点的类标为 D_i 中样本所对应的类标。
 - 如果 D_i 中的样本属于多个类, 选择一个特征及对应的判定条件将 D_i 分为两个或者多个子集; 为每个子集构建相应的子结点, 加入集合 N , 并将 D_i 中相应的数据赋给对应子结点; 以判定条件构造结点 i 到各子结点的边, 并加入集合 E 。
- (4) 重复执行步骤 3, 直到 N 中的所有结点都得到处理。此时结点集合 N 与边集合 E 联合构成所求决策树。

算法 6-1 比较粗略, 但基本上所有实际中的决策树算法都可以认为是算法 6-1 的细化版。在细化算法 6-1 的过程中, 从训练集生成决策树的两个关键问题如下:

- (1) 在构建新结点时如何选择特征及对应的判定条件;
- (2) 如何停止构建新结点? 或者换言之, 什么样的结点应该被认为是叶结点? 理想情况下, 所有样本的类别都一致时, 我们可以停止构建新结点, 但在实际情况中一般难以做到。

在下面的章节中, 我们将详细讨论这两个问题。对于第一个问题, 我们介绍 CART 中使用的基尼指数的增益和 C4.5 中使用的信息增益, 这里 CART 和 C4.5 是常用的两种商用决策树实现。对于第二个问题, 通常采取限制叶结点总数或叶结点所对应的训练集中样本数目的办法。在下面关于剪枝的讨论中, 我们将予以详细介绍。

1. 熵与基尼指数

如前所述, 在决策树的生成过程中, 我们实际上希望在划分树的结点所对应的样本集合后, 所得每个子集的“纯度”尽可能高, 尽量属于一个类。为了度量数据的“不纯程度”或“多样性”, 可引入信息论中熵的概念。对于一个给定的数据集, 我们可以根据集合中样本的类别来计算该数据集对应的熵。直观地讲, 我们想要用熵来帮助我们区分一个数据集是否是:

- (1) 已经完全分类好的数据集;
- (2) 已经近似分类好的数据集;
- (3) 还需要进一步分类的数据集。

首先考虑两类分类问题。严格地讲, 对于数据集 S , 其对应的熵的定义如下:

$$\text{Entropy}(S) = -P_+ \log_2 P_+ - P_- \log_2 P_- \quad (6-1)$$

这里 P_+ 是样本属于正类的概率, P_- 是样本属于负类的概率。特别地, 当 $P_+ = 0$ 或 $P_- = 0$ 时, 定义熵为 0。我们可以将上面熵的定义推广到多类问题中。假设有 c 个类, 使用 P_i 表示样本属于第 i 个类的概率, 则对应的熵可定义为:

$$\text{Entropy}(S) = -\sum_{i=1}^c P_i \log_2 P_i \quad (6-2)$$

图 6-2 中的实线给出了两类问题中熵的图像。其中 x 轴是 P_+ , y 轴是对应的熵。从图 6-2 中可以看出, 当 P_+ 的值为 0.5 时, 熵达到最大值 1。当 P_+ 从 0 增加到 0.5 时, 熵是单调递增的; 当 P_+ 从 0.5 增加到 1 时, 熵是单调递减的。

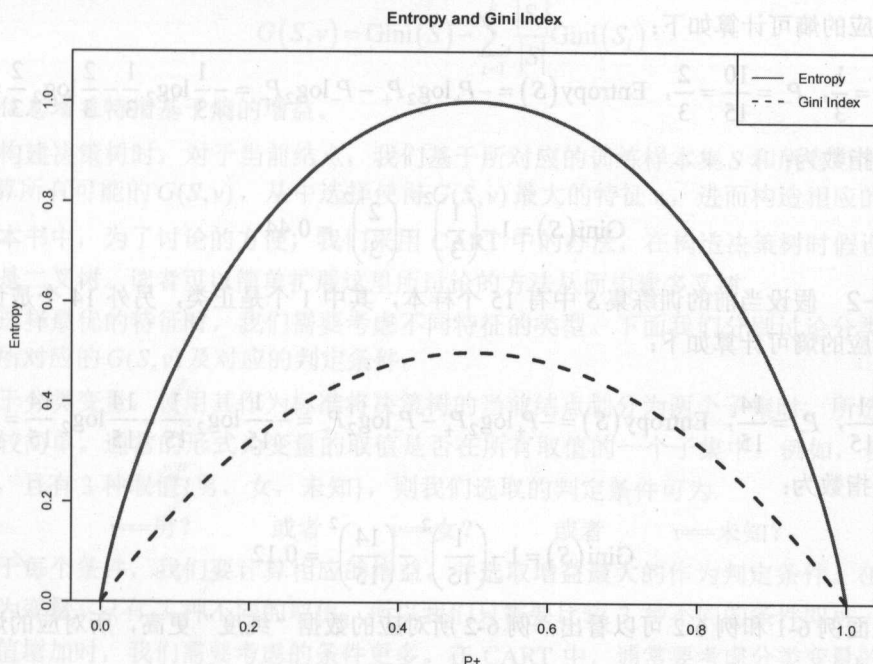


图 6-2 熵和基尼指数的变化曲线

根据熵的计算公式和对应图像可知: 当数据集中所有的样本点都属于同一类时, 所得的熵为 0; 在两类问题中, 当样本集中每类各占一半时, 熵取最大值 $\log_2 2 = 1.0$; 当某一类在样本集中占大多数时, 熵的值介于两者之间。

根据熵的定义, 在分类问题中, 我们需要使熵尽可能小。换言之, 就是使得数据的“纯度”尽可能高。在建立决策树时, 我们不断地建立新规则, 使得划分之后数据集的“纯度”更高, 即熵更小。因此, 从某种程度上讲, 我们可以把熵视为一种损失函数。

另外一种广泛使用的度量是基尼指数 (Gini index)。在两类分类问题中, 基尼指数定义为:

$$\text{Gini}(S) = 1 - P_+^2 - P_-^2 \quad (6-3)$$

在多元问题中, 对应的定义为:

$$\text{Gini}(S) = 1 - \sum_{i=1}^c P_i^2 \quad (6-4)$$

图 6-2 中的虚线给出了基尼指数的图像。从图 6-2 中可以看出, 基尼指数也是用于衡量数据“不纯程度”的度量。当数据来自同一类时, 基尼指数为 0; 当数据来自两类且各占一半时, 基尼指数达到最大值 0.5。

下面我们通过例 6-1 和例 6-2 来说明熵和基尼指数的计算。

例 6-1 假设当前的训练集 S 中有 15 个样本, 其中 5 个是正类, 另外 10 个是负类。该训练集对应的熵可计算如下:

$$P_+ = \frac{5}{15} = \frac{1}{3}, P_- = \frac{10}{15} = \frac{2}{3}, \text{Entropy}(S) = -P_+ \log_2 P_+ - P_- \log_2 P_- = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} = 0.92$$

基尼指数为:

$$\text{Gini}(S) = 1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = 0.44$$

例 6-2 假设当前的训练集 S 中有 15 个样本, 其中 1 个是正类, 另外 14 个是负类。该训练集对应的熵可计算如下:

$$P_+ = \frac{1}{15}, P_- = \frac{14}{15}, \text{Entropy}(S) = -P_+ \log_2 P_+ - P_- \log_2 P_- = -\frac{1}{15} \log_2 \frac{1}{15} - \frac{14}{15} \log_2 \frac{14}{15} = 0.35$$

基尼指数为:

$$\text{Gini}(S) = 1 - \left(\frac{1}{15}\right)^2 - \left(\frac{14}{15}\right)^2 = 0.12$$

从上面例 6-1 和例 6-2 可以看出, 例 6-2 所对应的数据“纯度”更高, 所对应的熵和基尼指数都更小。

2. 信息增益

根据熵的定义, 如果一个数据集的熵较大, 意味着该数据集对应的类别“纯度”较低, 需要继续划分该数据集。在建立决策树的过程中, 在每一步都需要选择分类能力最强的特征。简单地讲, 分类能力较强的特征能够使得将训练集划分成两个或者多个子集之后, 每个子集的“纯度”较高, 即熵较小。我们将划分前后熵的差称为由此特征划分所导致的信息增益 (information gain)。

严格地讲, 假设当前结点对应的训练集是 S , 我们使用特征 v 将 S 划分为 k 个不相交的子集 S_1, S_2, \dots, S_k

$$S = S_1 \cup S_2 \cup \dots \cup S_k \quad (6-5)$$

则使用特征 v 划分 S 的信息增益 $G(S, v)$ 计算公式如下:

$$G(S, v) = \text{Entropy}(S) - \sum_{i=1}^k \frac{|S_i|}{|S|} \text{Entropy}(S_i) \quad (6-6)$$

我们知道 $\text{Entropy}(S_i) \geq 0$, 因此 $G(S, v) \leq \text{Entropy}(S)$ 。根据信息增益的定义, 分类能力

越强的特征所对应的信息增益越大。在理想情况下, 如果在一个两类问题中, 特征 v 将 S 划分为 S_1 、 S_2 , 且每个子集中都只包含一类的样本, 即特征 v 能够完美地将 S 分类, 则 $G(S, v)$ 达到最大值 $\text{Entropy}(S)$ 。

在上面的讨论中, 我们可以将熵替换成基尼指数, 所有的结论仍然成立。对于基尼指数, 我们可以定义相应的增益如下:

$$G(S, v) = \text{Gini}(S) - \sum_{i=1}^k \frac{|S_i|}{|S|} \text{Gini}(S_i) \quad (6-7)$$

注意, 信息增益特指基于熵的增益。

在构建决策树时, 对于当前结点, 我们基于所对应的训练样本集 S 和所有可考虑的特征 v , 计算所有可能的 $G(S, v)$, 从中选择使得 $G(S, v)$ 最大的特征 v , 进而构造相应的子结点和边。在本书中, 为了讨论的方便, 我们采用 CART 中的办法, 在构造决策树时假设所得的决策树都是二叉树。读者可以简单扩展这里所讨论的方法从而构建多叉树。

在选择最优的特征时, 我们需要考虑不同特征的类型。下面我们分别讨论分类变量和数值变量所对应的 $G(S, v)$ 及对应的判定条件。

对于分类变量, 使用其作为标准将决策树的当前结点划分为两个子集时, 所选用的判定条件比较简单。通常的形式为变量的取值是否在所有取值的一个子集中。例如, 若变量 v 表示性别, 且有 3 种取值 {男, 女, 未知}, 则我们选取的判定条件可为

$$v = \text{男?} \quad \text{或者} \quad v = \text{女?} \quad \text{或者} \quad v = \text{未知?}$$

对于每个条件, 我们要计算相应的增益, 并选取增益最大的作为判定条件。在这个例子中, 因为变量 v 只有 3 种不同的取值, 所以我们只需要比较 3 种不同的条件即可。当变量的不同取值增加时, 我们需要考虑的条件更多。在 CART 中, 通常要考虑分类变量的所有可能的取值, 并从中选取最优的划分。

对于数值变量, 我们通常选取一个分割点, 并划分数据。例如, 对于数值变量 v , 我们可以使用条件 $v < t$ 来将数据分为两部分, 这里 t 就是分割点。在实际中, 对于数值变量 v , 我们可以搜集训练集中该变量的所有取值, 并按从小到大的顺序排列起来; 然后将相邻取值的算术平均值作为分割点, 并从中选出最优的分割点。具体来说, 假设数值变量 v 在训练集中的值按照从小到大的顺序排列为 $v_1 < v_2 < v_3 < \dots < v_n$, 则我们需要考虑如下不同的分割条件:

$$v < \frac{v_1 + v_2}{2}, v < \frac{v_2 + v_3}{2}, \dots, v < \frac{v_{n-1} + v_n}{2}$$

对于每个分割条件, 我们可以计算对应的增益, 并从中选出最优的划分。

下面我们通过例 6-3 来计算熵和相关特征的信息增益。

例 6-3 在本例中, 我们要找出表 6-4 中对应最大信息增益的变量及判定条件。表 6-4 中数据有两个特征, 即 color 和 weight; 从第二行起每行对应一个样本, 每列 (不包含最后一列) 对应一个特征; 最后一列对应类标 class (1 表示正类, 0 表示负类)。下面我们计算每

个变量及对应的判定条件, 根据最大信息增益的原则找出最优划分。

表 6-4 示例数据

color	weight	class
Red	100	1
Red	50	1
Blue	120	0
Green	80	0
Red	150	0
Green	90	1

变量 color 有 3 个不同的值 Red、Blue 和 Green, 对该变量我们需要考虑如下 3 个划分:

- color==Red 和 color!=Red
- color==Blue 和 color!=Blue
- color==Green 和 color!=Green

在划分之前的数据集里有 3 个正类样本、3 个负类样本, 因此, 划分之前熵为 $-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}=1.0$ 。考虑划分 color==Red 和 color!=Red。满足 color==Red 的有 3 个样本, 其中有 2 个正类、1 个负类, 对应的熵为 $-\frac{2}{3}\log_2\left(\frac{2}{3}\right)-\frac{1}{3}\log_2\left(\frac{1}{3}\right)=0.92$; 满足 color!=Red 的有 3 个样本, 其中有 1 个正类和 2 个负类, 对应的熵为 $-\frac{1}{3}\log_2\left(\frac{1}{3}\right)-\frac{2}{3}\log_2\left(\frac{2}{3}\right)=0.92$ 。因此, 划分 color==Red 和 color!=Red 对应的信息增益为:

$$1.0 - \left(\frac{3}{6} \times 0.92 + \frac{3}{6} \times 0.92 \right) = 1 - 0.92 = 0.08$$

下面考虑划分 color==Blue 和 color!=Blue。满足 color==Blue 的有 1 个负类样本, 对应的熵为 0; 满足 color!=Blue 的有 5 个样本, 其中 3 个正类、2 个负类, 对应的熵为 $-\frac{3}{5}\log_2\left(\frac{3}{5}\right)-\frac{2}{5}\log_2\left(\frac{2}{5}\right)=0.97$ 。因此, 划分 color==Blue 和 color!=Blue 对应的信息增益为:

$$1.0 - \left(\frac{1}{6} \times 0 + \frac{5}{6} \times 0.97 \right) = 0.19$$

类似地, 划分 color==Green 和 color!=Green 对应的信息增益为:

$$\begin{aligned} & 1.0 - \left(\frac{2}{6} \times \left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) \right) + \frac{4}{6} \times \left(-\frac{2}{4}\log_2\left(\frac{2}{4}\right) - \frac{2}{4}\log_2\left(\frac{2}{4}\right) \right) \right) \\ &= 1.0 - \left(\frac{1}{3} \times 1.0 + \frac{2}{3} \times 1.0 \right) = 0 \end{aligned}$$

因此, 对于变量 color, 最优的划分是 color==Blue 和 color!=Blue, 对应的信息增益

是 0.19。

接下来我们考虑第二个变量 weight 。将 weight 所有不同的取值排好序, 我们得到 50, 80, 90, 100, 120, 150。因此, 我们需要考虑如下划分:

- $\text{weight} < \frac{50+80}{2} = 65$
- $\text{weight} < \frac{80+90}{2} = 85$
- $\text{weight} < \frac{90+100}{2} = 95$
- $\text{weight} < \frac{100+120}{2} = 110$
- $\text{weight} < \frac{120+150}{2} = 135$

下面只讨论前面两个判定条件所对应的信息增益计算, 其他划分可以采用类似的方法计算。事实上, 有快速算法可以计算所有划分所对应的信息增益, 这里不详细讨论。

考虑判定条件 $\text{weight} < 65$ 。满足该条件的有 1 个正类样本, 所对应的熵为 0; 不满足该条件有 2 个正类、3 个负类样本, 所对应的熵为 $-\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right) = 0.97$ 。因此, 该判定条件所对应的信息增益为:

$$1.0 - \left(\frac{1}{6} \times 0 + \frac{5}{6} \times 0.97 \right) = 0.19$$

接下来考虑判定条件 $\text{weight} < 85$ 。满足该条件的有 1 个正类样本和 1 个负类样本, 所对应的熵为 $-\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1.0$; 不满足该条件的有 2 个正类样本、2 个负类样本, 所对应的熵为 $-\frac{2}{4}\log_2\left(\frac{2}{4}\right) - \frac{2}{4}\log_2\left(\frac{2}{4}\right) = 1.0$ 。因此, 该判定条件所对应的信息增益为:

$$1.0 - \left(\frac{2}{6} \times 1.0 + \frac{4}{6} \times 1.0 \right) = 0$$

表 6-5 列出了变量 weight 的所有判定条件所对应的信息增益。从表 6-5 中我们可以看出, 对于变量 weight , 最优的划分是 $\text{weight} < 110$, 对应的信息增益是 0.46。

表 6-5 特征 weight 不同的判定条件对应的信息增益

判定条件	信息增益
$\text{weight} < 65$	0.19
$\text{weight} < 85$	0
$\text{weight} < 95$	0.08
$\text{weight} < 110$	0.46
$\text{weight} < 135$	0.19

综合考虑两个变量可知，最优的划分是 $\text{weight} < 110$ 。

6.2.3 过拟合和剪枝

奥卡姆剃刀 (Occam's Razor, 也写作 Ockham's Razor) 是机器学习中广泛使用的一条规则, 由 14 世纪英格兰逻辑学家 William of Occam (约 1287—1347 年) 提出。简单地讲, 就是优先选择简单的模型。

对于同一组数据, 假设我们有两个分类性能相似的决策树模型, 那么, 在实际中我们应该优先选择更简单的模型。在第 5 章回归算法中, 我们讨论了算法的复杂度。在决策树中, 通常我们可以使用叶结点的数目或者决策树的深度来描述决策树模型的复杂度。决策树本身的特点决定了它比较容易导致过拟合。在这种情况下, 通过剪枝 (pruning) 来控制模型的复杂度是一种非常有效和必要的手段。

在决策树中, 有两种剪枝方法来控制决策树的复杂度:

- 事前剪枝 (pre-pruning);
- 事后剪枝 (post-pruning)。

事前剪枝就是在决策树的生成过程中设定一些指标条件, 用来决定每个结点是否应该继续划分。具体来说, 就是在决策树的生成过程中, 如果某一结点对应的统计指标达到某一阈值, 则停止划分; 否则继续划分该结点。在实际中使用的指标条件包括:

- 该结点对应的训练样本数目低于某一阈值;
- 继续划分虽然导致熵或者基尼指数降低, 但降低量低于某一阈值。

事前剪枝能够避免生成过于复杂的决策树, 计算复杂度较低, 但也有“近视”的缺点。在一些实际的决策树生成过程中, 有可能对当前结点进一步划分不能提高分类效果, 但是在下一层再划分却有显著的效果。

事后剪枝就是先让决策树充分生长, 然后自底向上进行剪枝, 以降低模型的复杂度。一般而言, 我们可以将一棵子树用一个叶结点取代 (叶结点的类标由对应训练集中样本数最多的类标决定)。与事前剪枝相比, 事后剪枝可以避免近视的弱点, 但计算复杂度更高。

下面我们介绍 R 中的 rpart 软件包^①如何剪枝。首先, 在使用 rpart 函数构建决策树时, 我们可以使用如下 3 个参数来控制决策树的生成, 属于事前剪枝的范畴。

- `minsplit`: 在生成决策树的过程中, 该结点对应的训练集样本数必须超过 `minsplit` 值才能考虑进一步划分。
- `minbucket`: 叶结点所对应的训练样本最小数目。
- `maxdepth`: 决策树的最大深度。

在得到相应的决策树之后, 我们可以进一步使用交叉检验 (参见 6.6.1 节) 来对所得的决策树进行事后剪枝。rpart 中有一个参数 `cp` 可以用于控制模型的复杂度, 设置不同的 `cp` 值可以得到结点数目差异很大的决策树。这里 `cp` 是 Complexity Parameter 的简写。我们可以使

^① <https://cran.r-project.org/web/packages/rpart/index.html>

用多个 cp 值构建相应的决策树，并利用交叉检验计算所得模型在检验集上的错误，从而选取最优的 cp 值并利用其剪枝。

6.2.4 实际使用

在前面的章节中我们讨论的都是关于决策树的一些通用的技术。在具体的实现中，有两种较知名的决策树实现：

- CART (classification and regression tree)

- C4.5

这两种算法都实现了决策树，但在一些具体实现上有差别。

- 在 CART 中，得到的决策树是二叉树，而在 C4.5 中，可以得到多叉树。
- 标准的 CART 算法使用基尼指数来确定最优的特征和对应的分割点，而 C4.5 使用熵和信息增益。
- 两种算法的剪枝方法不同。
- 对于缺失值的处理不同。

R 中的 `rpart` 软件包主要基于 CART 算法实现，但在很多地方做了扩充。这里包名 `rpart` 是 recursive partitioning 的简写。例如，在 `rpart` 中我们可以选择使用基尼指数的增益或者信息增益来选择下一步的最优特征和对应的分割点。与 CART 相同，在 `rpart` 中我们只能得到二叉树。

在 `rpart` 包中，我们使用同名函数 `rpart` 构建决策树。在构建决策树时，主要参数包括 `cp`、`minsplit`、`minbucket` 和 `maxdepth`，其含义在前面的剪枝部分已经予以介绍。

与回归问题类似，对于当前的决策树，我们可以在训练集上定义并计算错误率。在决策树的生成过程中，虽然划分结点可以降低错误率，但同时也增加了模型的复杂度。在 `rpart` 中，我们使用参数 `cp` 来控制模型在训练集上的错误率和模型的结点数目之间的关系。在 `rpart` 中， cp 值一般在 0 和 1 之间。当 $cp=1$ 时，我们得到只有一个根结点的决策树（模型复杂度最低，但在训练集上的分类结果不理想）；当 $cp=0$ 时，我们在训练时不控制结点的数目（模型复杂度很高，但在训练集上的分类结果很理想）。通常我们会尝试多个 cp 值并通过交叉检验得到最优的 cp 值。

使用 `rpart` 函数时，我们先利用 `rpart.control` 函数将上面所讨论的控制参数放到一起，再调用 `rpart` 函数构建决策树。`rpart` 函数的用法如下：

```
rpart(formula, data, weights, subset, na.action = na.rpart, method,
      model = FALSE, x = FALSE, y = TRUE, parms, control, cost, ...)
```

其中第一个参数 `formula` 是公式，表示在给定的数据中，哪一列对应类标 y ，哪些列对应 x ；第二个参数 `data` 表示输入数据；参数 `weights`、`subset`、`na.action`、`model`、`x`、`y`、`parms`、`cost` 等都是可选的；参数 `method` 表示生成决策树的类型，最常用的取值包括 'class' 和 'anova'，分别表示构建分类树和回归树；参数 `control` 的值可以使用上面讨论的 `rpart.control` 传入。

在得到训练所得的决策树后，可以使用 `printcp` 函数打印多个 cp 值对应的交叉检验中

训练集和检验集上的错误。根据交叉检验的结果，我们可以选取最优的 cp 值。

在得到一个决策树后，可以使用 `prune` 函数剪枝。在使用 `prune` 时，第一个参数就是已有的 `rpart` 生成的决策树对象，第二个参数是我们选择的 cp 值。利用 `printcp` 的输出，我们可以选出使得检验集分类最优的 cp 值，并利用该 cp 值对所得决策树剪枝。

在下面的讨论中，我们使用实际的数据和例子来说明如何使用 `rpart` 构建决策树。这里我们着重讲解如何使用 `rpart` 来生成不同的决策树、如何剪枝、如何使用决策树来处理新的测试数据等，完整的 R 代码在文件 `rpart_example_kyphosis.R` 中。

```
# Step 0. Check required packages are installed or not. If not, install them.
rpart.installed <- 'rpart' %in% rownames(installed.packages())
if (rpart.installed) {
  print("the rpart package is already installed, let's load it...")
}else {
  print("let's install the rpart package first...")
  install.packages('rpart', dependencies=T)
}
library('rpart')

# The partykit is also used to draw the decision tree
# partykit provides some very good graphing and visualization of tree models
partykit.installed <- 'partykit' %in% rownames(installed.packages())
if (partykit.installed) {
  print("the partykit package is already installed, let's load it...")
}else {
  print("let's install the partykit package first...")
  install.packages('partykit', dependencies=T)
}
library('partykit')

#=====
# Step 1. Load the data
# load the data
data(kyphosis)
D <- kyphosis
print('the basic information of D')
str(D)

# Step 2. Train decision tree models using rpart
# Step 2.1 Train a rpart model using default settings
M_rpart1 <- rpart(Kyphosis~., data = D, method = 'class')
print('show the trained model')
print(M_rpart1)
print('show the detailed summary of splits')
summary(M_rpart1)
print('plot the tree')
plot(as.party(M_rpart1), main='decision tree using default setting')
```

```

# Step 2.2 Check cp and the corresponding CV error
# 'cp' stands for Complexity Parameter of the tree.
# plot a complexity parameter table for an Rpart Fit
plotcp(M_rpart1)
# display CP table for fitted Rpart object
C <- printcp(M_rpart1)

# Step 2.3. Prune the resulting decision using the optimal cp value and the
# prune function
cp_optimal <- C[which.min(C[, 'xerror']), 'CP']
M_rpart1.1 <- prune(M_rpart1, cp = cp_optimal)
print(M_rpart1.1)
print('plot the pruned tree')
plot(as.party(M_rpart1.1), main='pruned decision tree')

# Step 2.4 Train a tree using more complicated parameter setting
M_rpart2 <- rpart(Kyphosis~., data = D, method = 'class',
                  control = rpart.control(minsplit = 10, minbucket = 5, maxdepth = 4))
print('show the summary of the trained model')
print(M_rpart2)
# plot the resulting tree
plot(as.party(M_rpart2), main='decision tree using advanced parameter setting')

# Step 3. Make prediction using the predict function, and compute accuracy
# We use the first 10 samples from D as the test data set
D_test <- D[1:10,]
# Output the probability first
y_test_prob <- predict(M_rpart1, D_test)
print(y_test_prob)
# Get the explicit class label next
y_test_label <- predict(M_rpart1, D_test, type='class')
print(y_test_label)
# Compute the corresponding accuracy
accuracy_test <- sum(D_test$Kyphosis==y_test_label) / length(y_test_label)
print(paste0('accuracy on the test data set is ', accuracy_test))

```

这段程序首先检查 rpart 和 partykit 软件包有没有安装。如果没有安装的话，则要安装它们。这里我们使用 partykit 软件包来可视化生成的决策树，因此也要预先安装 partykit 包。事实上，rpart 软件包也提供了可视化决策树的函数，但在显示效果上不如 partykit 包。我们稍后会详细介绍多种可视化决策树的方法。

rpart 包中包含了 kyphosis 数据集。我们可以直接使用 data(kyphosis) 导入该数据集（得到一个名为 kyphosis 的数据框），并将其赋给变量 D。在 R 中，我们可以使用 str 函数来显示任何 R 对象的主要信息。下面是 str(D) 的输出：

```

[1] "the basic information of D"
'data.frame':81 obs. of 4 variables:

```

```
$ Kyphosis: Factor w/ 2 levels "absent","present": 1 1 2 1 1 1 1 1 2 ...
$ Age      : int   71 158 128 2 1 1 61 37 113 59 ...
$ Number   : int   3 3 4 5 4 2 2 3 2 6 ...
$ Start    : int   5 14 5 1 15 16 17 16 16 12 ...
```

从以上输出可以看出，D有4列，分别是 Kyphosis、Age、Number 和 Start。这组数据集对应一个分类问题，其中的类标是 Kyphosis 列，有两个不同的值 absent 和 present 来表示不同的类标。这里 Kyphosis 是“驼背”的意思，我们要用其他的3个变量来预测它。

在上面的 R 代码中，我们构建了多个不同的决策树模型。在第一个模型 M_rpart1 中，我们直接使用了默认的参数配置来构建决策树：

```
M_rpart1 <- rpart(Kyphosis~., data = D, method = 'class')
```

这里第一个参数是一个公式对象，我们使用 Kyphosis~. 表示需要预测 Kyphosis，并使用所有的其他变量来进行预测。此处将 method 参数设置为 'class' 表示要构建一个适用于分类问题的决策树。

使用 print 函数可以直接打印生成的决策树 M_rpart1 的主要信息。这里其对应的输出如下：

```
n= 81
node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 81 17 absent (0.79012346 0.20987654)
  2) Start>=8.5 62 6 absent (0.90322581 0.09677419)
    4) Start>=14.5 29 0 absent (1.00000000 0.00000000) *
    5) Start< 14.5 33 6 absent (0.81818182 0.18181818)
      10) Age< 55 12 0 absent (1.00000000 0.00000000) *
      11) Age>=55 21 6 absent (0.71428571 0.28571429)
        22) Age>=111 14 2 absent (0.85714286 0.14285714) *
        23) Age< 111 7 3 present (0.42857143 0.57142857) *
      3) Start< 8.5 19 8 present (0.42105263 0.57894737) *
```

可以看出，print(M_rpart1) 将生成的决策树以文本的形式打印出来，并打印了每个结点所对应的训练集的分布情况。我们还可以使用 summary 函数打印更详细的信息，这里就不列出了。读者可以直接运行我们提供的 R 程序并查看相应的输出。

由前面的讨论可以看出使用 rpart 构建决策树时参数 cp 的重要性。我们可以使用 printcp 打印出不同 cp 取值下，所构建的决策树 M_rpart1 在交叉检验中的训练集和检验集上的错误率等信息。下面给出了 printcp(M_rpart1) 的输出，从中可以看到决策树中仅使用了两个变量。在最后打印出的表中，显示了不同的 cp 值、对应分割结点数 nsplit（即决策树中非叶结点的数目，决策树的总叶结点数是该值加1）以及所得决策树在训练集上的错误率（列 relerror）、在检验集上的错误率（列 xerror）、在检验集上错误率的标准差（列 xstd）。注意，在 relerror、xerror 中我们都使用了相对的错误率。从打印结果可以知道，根结点对应的错误率是 17/81=0.20988。同时我们也知道当 cp=0.019608 时，对应的 xerror=0.82353

是相对于根结点的错误率的比值,此时真正的错误率应该是 $0.82353 \times 0.20988 = 0.1728425$ 。由此也可以看出,当 cp 值小于 0.019608 之后,或者决策树的总结点数大于或等于 2 之后,在检验集上的错误率都没有变化。

Classification tree:

```
rpart(formula = Kyphosis ~ ., data = D, method = "class")
```

Variables actually used in tree construction:

```
[1] Age Start
```

Root node error: 17/81 = 0.20988

n= 81

	CP	nsplit	rel error	xerror	xstd
1	0.176471	0	1.00000	1.00000	0.21559
2	0.019608	1	0.82353	0.88235	0.20565
3	0.010000	4	0.76471	0.88235	0.20565

`plotcp` 函数经常与 `printcp` 一起使用,它以图像的形式呈现了 cp 值与 $xerror$ 的关系。图 6-3 给出了 `plotcp` 函数的输出。

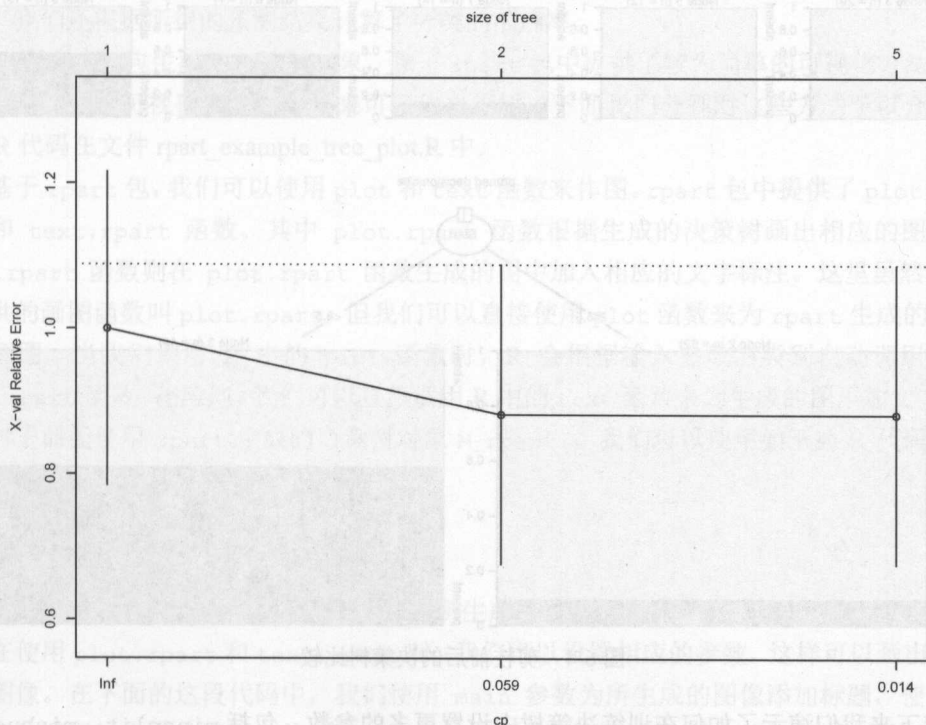


图 6-3 在不同 cp 值检验集上错误率 ($xerror$) 的变化 ($size$ of tree 是 cp 值对应的决策树的总结点数)

这里我们选取最优 cp 值的标准是使得 $xerror$ 最小，同时决策树的结点总数最小。我们在代码中将最优结果保存在 $cp_optimal$ 中，在所用数据集上 $cp_optimal=0.01960784$ 。这样我们就可以直接使用 `prune` 函数对决策树模型 M_rpart1 进行剪枝，得到模型 $M_rpart1.1$ ：

```
M_rpart1.1 <- prune(M_rpart1, cp = cp_optimal)
```

相比 M_rpart1 ，新的决策树模型 $M_rpart1.1$ 更加简单。在上面的 R 程序中，我们使用 `plot(as.party(M_rpart1))` 和 `plot(as.party(M_rpart1.1))` 来可视化剪枝前后的两个决策树模型，结果如图 6-4 所示。

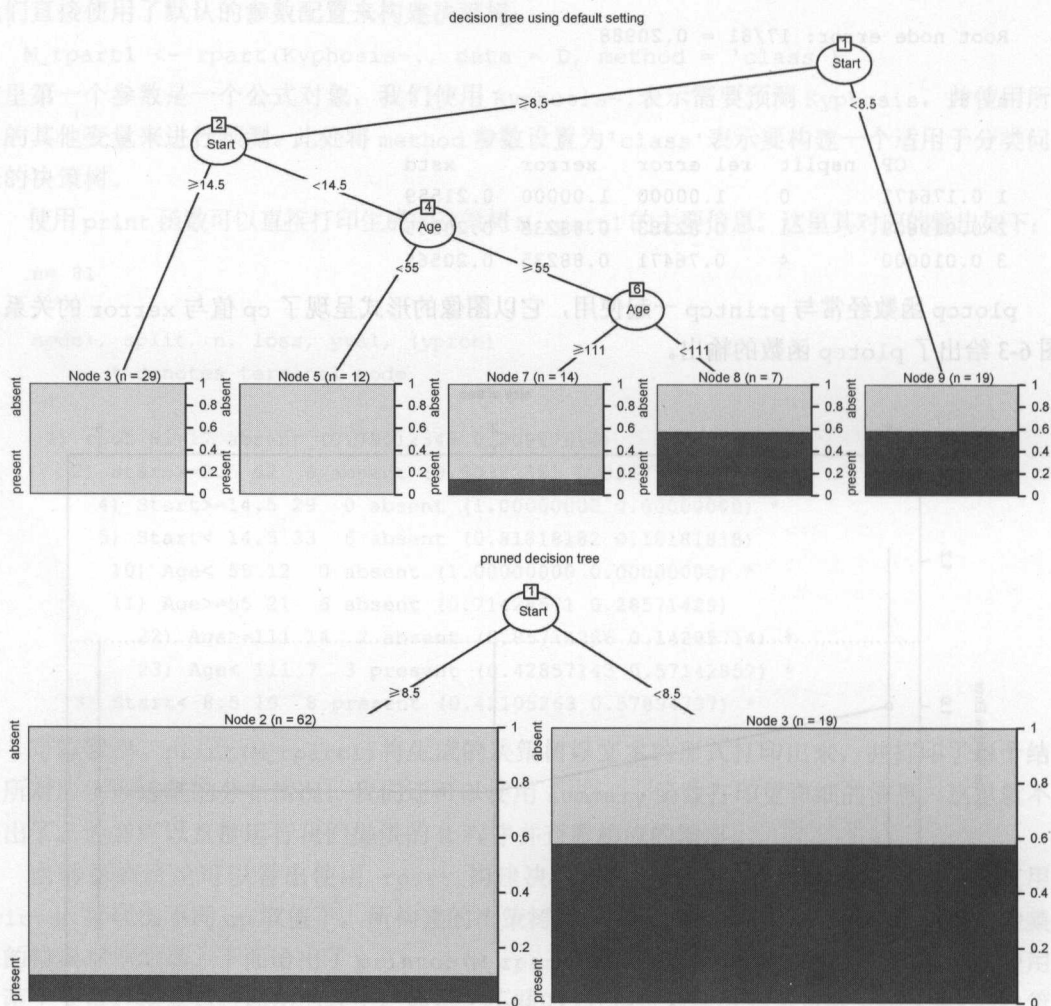


图 6-4 剪枝前后的决策树比较

接下来我们演示了如何在训练决策树中设置更多的参数，包括 `minsplit`、`minbucket`、`maxdepth` 等。

最后我们使用 `predict` 函数来预测新的样本。在 `predict` 函数中，我们要注意参数 `type`。当 `type='prob'` 时，`predict` 函数输出每个样本属于每个类的概率；当 `type` 的值设定为 `'class'` 时，`predict` 函数直接输出所得的类标。对于分类树来说，`type` 的默认值是 `'prob'`。因此，上面程序中两个 `predict` 函数的输出如下：

```
> print(y_test_prob)
      absent  present
1  0.4210526 0.5789474
2  0.8571429 0.1428571
3  0.4210526 0.5789474
4  0.4210526 0.5789474
5  1.0000000 0.0000000
6  1.0000000 0.0000000
7  1.0000000 0.0000000
8  1.0000000 0.0000000
9  1.0000000 0.0000000
10 0.4285714 0.5714286
> print(y_test_label)
      1      2      3      4      5      6      7      8      9     10
present absent present present absent absent absent absent absent present
Levels: absent present
```

此外，我们还根据所得的预测结果计算了分类的准确率。

对于 `rpart` 包生成的决策树对象，除了 `rpart` 包中提供了较为简单的可视化方法外，还有多个包提供了可视化相应的方法来可视化决策树。下面我们分别对这些方法予以介绍。完全的 R 代码在文件 `rpart_example_tree_plot.R` 中。

基于 `rpart` 包，我们可以使用 `plot` 和 `text` 函数来作图。`rpart` 包中提供了 `plot.rpart` 函数和 `text.rpart` 函数。其中 `plot.rpart` 函数根据生成的决策树画出相应的图像；而 `text.rpart` 函数则在 `plot.rpart` 函数生成的图中加入相应的文字标注。这里虽然 `rpart` 中提供的画图函数叫 `plot.rpart`，但我们可以直接使用 `plot` 函数来为 `rpart` 生成的决策树对象画图。当我们调用 R 中的 `plot` 函数时，R 会根据输入参数的类别自动调用对应的 `plot.rpart` 函数。相应地，我们可以直接调用 R 中的 `text` 函数来为生成的图添加文字注释。

对于前面使用 `rpart` 生成的决策树对象 `M_rpart1`，我们可以使用如下的 R 代码来作图并标注：

```
plot(M_rpart1)
text(M_rpart1)
```

生成的决策树图像如图 6-5 所示，可以看出所生成的图像不是特别理想。

在使用 `plot.rpart` 和 `text.rpart` 时，我们可以设置相应的参数，这样可以画出更加满意的图像。在下面的这段代码中，我们使用 `main` 参数为所生成的图像添加标题，使用 `cex` 参数调整所添加文字的比例，使用 `uniform` 参数使得决策树各层结点之间的距离一致。所生成的决策树图像如图 6-6 所示。

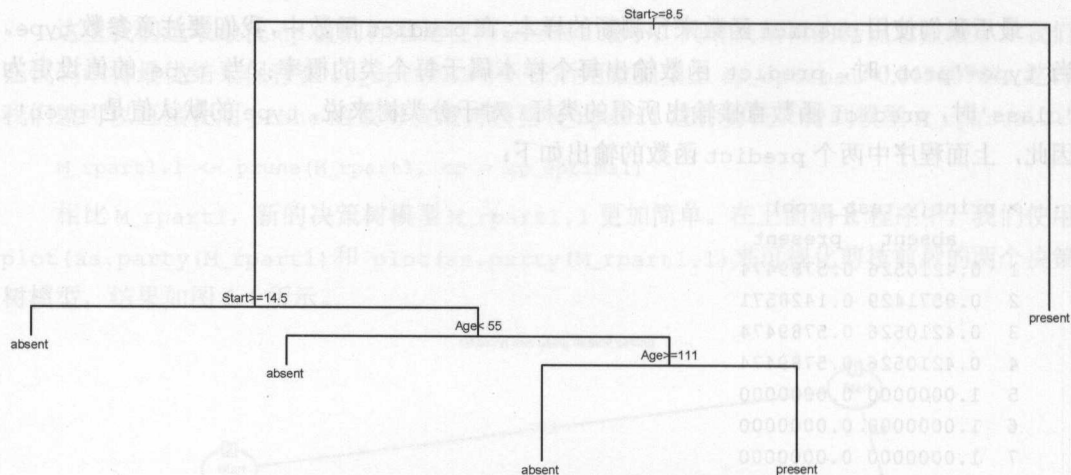


图 6-5 使用 rpart 中提供的函数生成的决策树图像

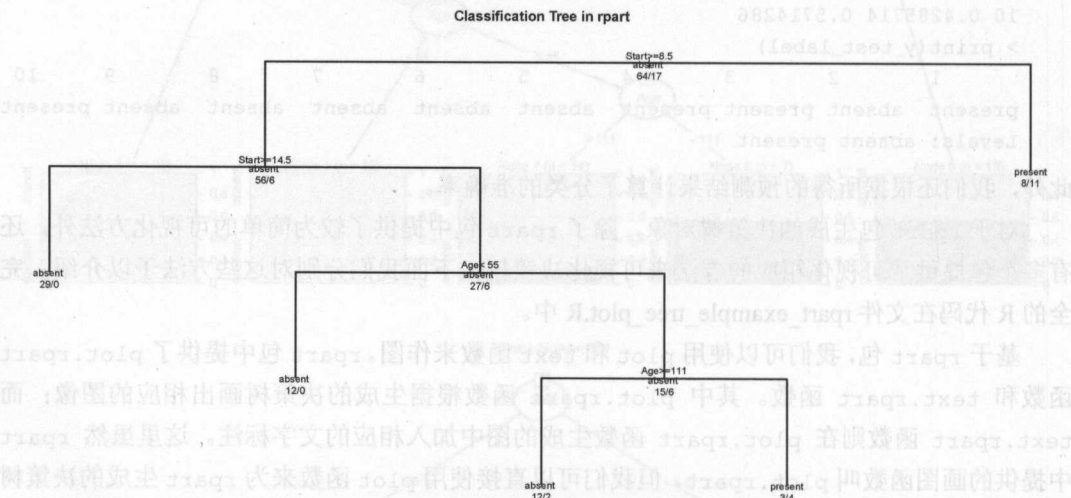


图 6-6 进一步调整参数所生成的决策树图像

```
plot(M_rpart1, uniform=TRUE, main="Classification Tree in rpart", cex = 0.5)
text(M_rpart1, use.n=TRUE, all=TRUE, cex=0.8)
```

此外,我们还可以使用 `rpart.plot` 包中的同名函数来可视化 `rpart` 生成的决策树。注意,这里也要首先安装 `rpart.plot` 包。使用下面的 R 代码,我们可以生成如图 6-7 所示的决策树图像。

```
rpart.plot(M_rpart1, main='Decision Tree in rpart.plot')
```

我们还可以使用 `rattle` 软件包来可视化 `rpart` 中生成的决策树。`rattle` 软件包依赖的软件包较多,需要较长时间下载和安装。安装好 `rattle` 软件包之后,我们可以使用其中的 `fancyRpartPlot` 函数来生成相应的图像:

```
fancyRpartPlot(M_rpart1, main='Decision Tree in rattle')
```

所生成的图像如图 6-8 所示。

Decision Tree in rpart.plot

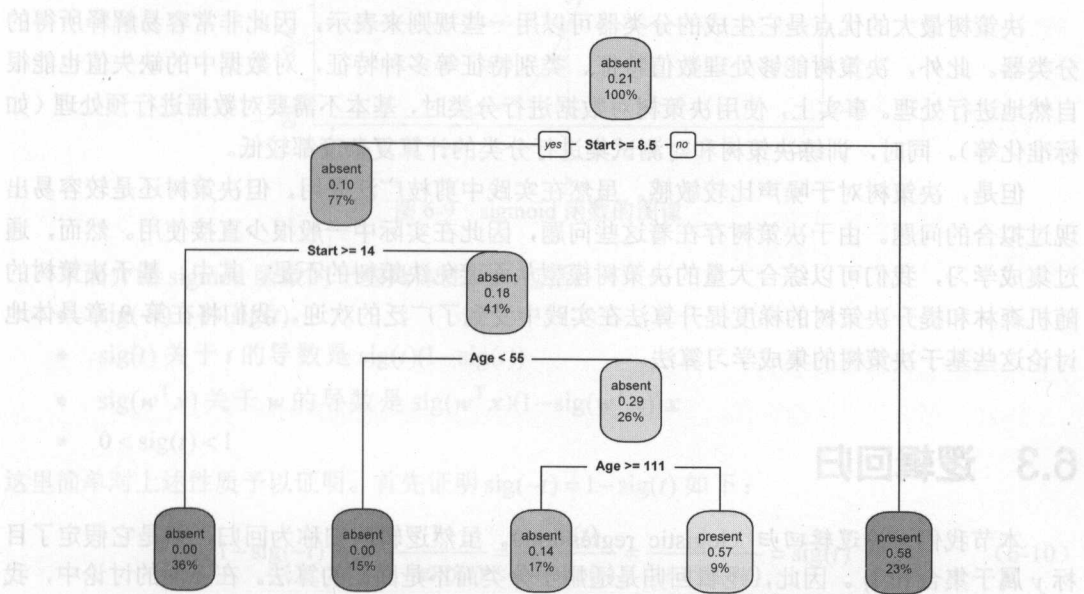
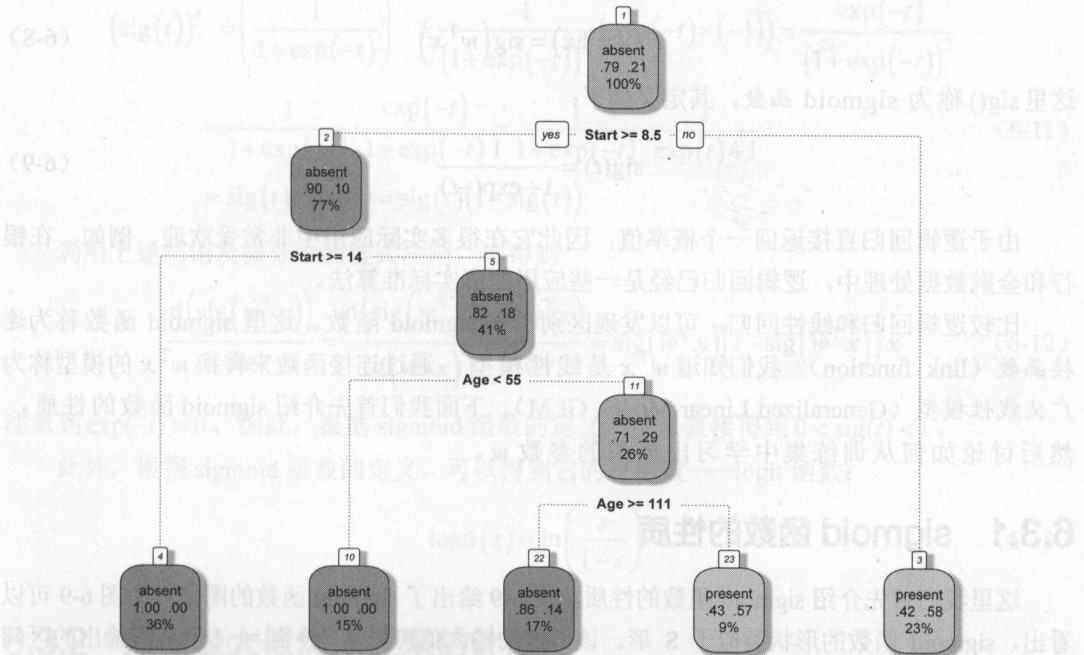


图 6-7 使用 `rpart.plot` 包生成的决策树图像

Decision Tree in rattle



Rattle 2016-Dec-18 20:00:28 Isun

图 6-8 使用 `rattle` 软件包生成的决策树图像

6.2.5 讨论

决策树最大的优点是它生成的分类器可以用一些规则来表示,因此非常容易解释所得的分类器。此外,决策树能够处理数值特征、类别特征等多种特征,对数据中的缺失值也能很自然地进行处理。事实上,使用决策树对数据进行分类时,基本不需要对数据进行预处理(如标准化等)。同时,训练决策树和对测试集进行分类的计算复杂度都较低。

但是,决策树对于噪声比较敏感。虽然在实践中剪枝广泛采用,但决策树还是较容易出现过拟合的问题。由于决策树存在着这些问题,因此在实际中一般很少直接使用。然而,通过集成学习,我们可以综合大量的决策树模型从而避免决策树的不足。其中,基于决策树的随机森林和提升决策树的梯度提升算法在实践中受到了广泛的欢迎。我们将在第9章具体地讨论这些基于决策树的集成学习算法。

6.3 逻辑回归

本节我们讨论逻辑回归(logistic regression)。虽然逻辑回归称为回归,但是它假定了目标 y 属于集合 $\{0,1\}$ 。因此,逻辑回归是适用于分类而不是回归的算法。在下面的讨论中,我们假设要解决的是两类分类问题,并将正类记为1,负类记为0。

在逻辑回归中,一个关键假设是样本 \mathbf{x} 属于正类的概率可以用下面的式子来表示:

$$P(y=1|\mathbf{x}) = \text{sig}(\mathbf{w}^T \mathbf{x}) \quad (6-8)$$

这里 $\text{sig}()$ 称为sigmoid函数,其定义如下:

$$\text{sig}(t) = \frac{1}{1 + \exp(-t)} \quad (6-9)$$

由于逻辑回归直接返回一个概率值,因此它在很多实际应用中非常受欢迎。例如,在银行和金融数据处理中,逻辑回归已经是一些应用的事实标准算法。

比较逻辑回归和线性回归,可以发现区别在于sigmoid函数。这里sigmoid函数称为连接函数(link function)。我们知道 $\mathbf{w}^T \mathbf{x}$ 是线性模型,通过连接函数来转换 $\mathbf{w}^T \mathbf{x}$ 的模型称为广义线性模型(Generalized Linear Model, GLM)。下面我们首先介绍sigmoid函数的性质,然后讨论如何从训练集中学习出模型的参数 \mathbf{w} 。

6.3.1 sigmoid函数的性质

这里我们首先介绍sigmoid函数的性质。图6-9给出了sigmoid函数的图像。由图6-9可以看出,sigmoid函数的形状类似于S形。该函数的输入值可以从 $-\infty$ 到 ∞ ,但是其输出的区间是 $(0,1)$ 。这样对于任意值的 $\mathbf{w}^T \mathbf{x}$,通过sigmoid函数,我们都能够得到0到1之间的概率。

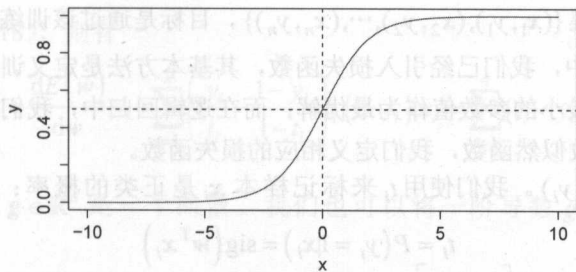


图 6-9 sigmoid 函数的图像

下面介绍 sigmoid 函数的一些具体性质, 包括:

- $\text{sig}(-t) = 1 - \text{sig}(t)$
- $\text{sig}(t)$ 关于 t 的导数是 $\text{sig}(t)(1 - \text{sig}(t))$
- $\text{sig}(\mathbf{w}^T \mathbf{x})$ 关于 \mathbf{w} 的导数是 $\text{sig}(\mathbf{w}^T \mathbf{x})(1 - \text{sig}(\mathbf{w}^T \mathbf{x}))\mathbf{x}$
- $0 < \text{sig}(t) < 1$

这里简单对上述性质予以证明。首先证明 $\text{sig}(-t) = 1 - \text{sig}(t)$ 如下:

$$1 - \text{sig}(-t) = 1 - \frac{1}{1 + \exp(t)} = \frac{\exp(t)}{1 + \exp(t)} = \frac{1}{1 + \exp(-t)} = \text{sig}(t) \quad (6-10)$$

接下来推导 $\text{sig}(t)$ 关于 t 的导数:

$$\begin{aligned} (\text{sig}(t))' &= \left(\frac{1}{1 + \exp(-t)} \right)' = \frac{-1}{(1 + \exp(-t))^2} (\exp(-t) \times (-1)) = \frac{\exp(-t)}{(1 + \exp(-t))^2} \\ &= \frac{1}{1 + \exp(-t)} \cdot \frac{\exp(-t)}{1 + \exp(-t)} = \frac{1}{1 + \exp(-t)} \cdot \frac{1}{\exp(t) + 1} \\ &= \text{sig}(t) \text{sig}(-t) = \text{sig}(t)(1 - \text{sig}(t)) \end{aligned} \quad (6-11)$$

利用上述结论及微分中的链式法则, 可得到

$$\frac{d(\text{sig}(\mathbf{w}^T \mathbf{x}))}{d\mathbf{w}} = \frac{d(\text{sig}(\mathbf{w}^T \mathbf{x}))}{d(\mathbf{w}^T \mathbf{x})} \frac{d(\mathbf{w}^T \mathbf{x})}{d\mathbf{w}} = \text{sig}(\mathbf{w}^T \mathbf{x})(1 - \text{sig}(\mathbf{w}^T \mathbf{x}))\mathbf{x} \quad (6-12)$$

注意到 $\exp(-t) > 0$, 因此, 根据 sigmoid 函数的定义, 可以直接得到 $0 < \text{sig}(t) < 1$ 。

此外, 根据 sigmoid 函数的定义, 可以得到它的反函数——logit 函数:

$$\text{logit}(s) = \ln \left(\frac{s}{1-s} \right) \quad (6-13)$$

6.3.2 通过极大似然估计来估计参数

在逻辑回归中唯一的参数是 \mathbf{w} 。本节讨论如何通过训练集使用极大似然估计来估计该参数。

假设我们有训练集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ ，目标是通过该训练集学习出逻辑回归的参数 \mathbf{w} 。在回归分析中，我们已经引入损失函数，其基本方法是定义训练集上的损失函数，并找出使得损失函数最小的参数值作为最优解；而在逻辑回归中，我们引入极大似然估计并找出最优解。根据对数似然函数，我们定义相应的损失函数。

首先考虑样本 (\mathbf{x}_i, y_i) 。我们使用 t_i 来标记样本 \mathbf{x}_i 是正类的概率：

$$t_i = P(y_i = 1 | \mathbf{x}_i) = \text{sig}(\mathbf{w}^T \mathbf{x}_i) \quad (6-14)$$

于是，给定 \mathbf{x}_i 时 y_i 是 0 的概率为 $P(y_i = 0 | \mathbf{x}_i) = 1 - P(y_i = 1 | \mathbf{x}_i) = 1 - t_i$ 。综合考虑这两种情况，我们可以将 y_i 的似然函数写为：

$$P(y_i | \mathbf{w}) = t_i^{y_i} (1 - t_i)^{(1 - y_i)} \quad (6-15)$$

注意，由于 y_i 只能是 0 或者 1，因此该公式始终只有一项。

因此，对于整个训练集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ 而言，似然函数可以写为：

$$P(\mathbf{y} | \mathbf{w}) = \prod_{i=1}^n t_i^{y_i} (1 - t_i)^{(1 - y_i)} \quad (6-16)$$

这里 $\mathbf{y} = (y_1, y_2, \dots, y_n)^T \in \mathbb{R}^n$ 。对于给定的训练集，我们可以最大化似然 $P(\mathbf{y} | \mathbf{w})$ ，从而得到最优的 \mathbf{w} 。在机器学习中，通常最小化损失函数。在逻辑回归中，引入一个新的损失函数，称为交叉熵（cross-entropy）损失函数：

$$E(\mathbf{w}) = -\ln P(\mathbf{y} | \mathbf{w}) = -\sum_{i=1}^n (y_i \ln t_i + (1 - y_i) \ln (1 - t_i)) \quad (6-17)$$

可以看出，交叉熵损失函数实际上就是似然函数先取对数再取相反数的结果。因此，最大化似然函数等价于最小化交叉熵损失函数。

在数值优化中，假设我们要最小化函数 f ，通常会利用其导数提供的信息。例如，在每一步我们可以沿着负导数的方向搜索，这样就可以在优化的每一步降低函数 f 的值。在最小化交叉熵损失函数时，我们可以计算交叉熵损失函数对于参数 \mathbf{w} 的导数 \mathbf{g} 和二阶导数 \mathbf{H} 。

首先，我们可以求 $E(\mathbf{w})$ 关于 \mathbf{w} 的导数 \mathbf{g} ，如下：

$$\begin{aligned} \mathbf{g} = \frac{dE(\mathbf{w})}{d\mathbf{w}} &= -\sum_{i=1}^n \left(y_i \frac{1}{t_i} \frac{dt_i}{d\mathbf{w}} + (1 - y_i) \frac{1}{1 - t_i} \frac{d(1 - t_i)}{d\mathbf{w}} \right) \\ &= -\sum_{i=1}^n \left(\frac{y_i}{t_i} \frac{dt_i}{d\mathbf{w}} - \frac{1 - y_i}{1 - t_i} \frac{dt_i}{d\mathbf{w}} \right) = -\sum_{i=1}^n \left(\frac{y_i}{t_i} - \frac{1 - y_i}{1 - t_i} \right) \frac{dt_i}{d\mathbf{w}} \end{aligned} \quad (6-18)$$

利用前面的性质，我们有：

$$\frac{dt_i}{d\mathbf{w}} = \frac{d(\text{sig}(\mathbf{w}^T \mathbf{x}_i))}{d\mathbf{w}} = t_i (1 - t_i) \mathbf{x}_i \quad (6-19)$$

将其代入式 (6-18), 则有

$$\mathbf{g} = \frac{dE(\mathbf{w})}{d\mathbf{w}} = -\sum_{i=1}^n \left(\frac{y_i}{t_i} - \frac{1-y_i}{1-t_i} \right) t_i (1-t_i) \mathbf{x}_i = \sum_{i=1}^n (t_i - y_i) \mathbf{x}_i \quad (6-20)$$

注意, 这里导数 $\mathbf{g} \in \mathbb{R}^d$ 是一个向量。我们也可以将一阶导数 \mathbf{g} 写成矩阵的形式:

$$\mathbf{g} = \frac{dE(\mathbf{w})}{d\mathbf{w}} = \sum_{i=1}^n (t_i - y_i) \mathbf{x}_i = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n] \begin{bmatrix} t_1 - y_1 \\ t_2 - y_2 \\ \vdots \\ t_n - y_n \end{bmatrix} = \mathbf{X}^T (\mathbf{t} - \mathbf{y}) \quad (6-21)$$

这里矩阵 $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times d}$ 为数据矩阵, $\mathbf{y} = [y_1, y_2, \dots, y_n]^T \in \mathbb{R}^n$ 为对应的类别信息, $\mathbf{t} = [t_1, t_2, \dots, t_n]^T \in \mathbb{R}^n$ 为逻辑回归所构建的模型的预测值。

类似地, 我们可以求得二阶导数 $\mathbf{H} \in \mathbb{R}^{d \times d}$:

$$\mathbf{H} = \sum_{i=1}^n t_i (1-t_i) \mathbf{x}_i \mathbf{x}_i^T = \mathbf{X}^T \mathbf{S} \mathbf{X} \quad (6-22)$$

这里矩阵 $\mathbf{S} \in \mathbb{R}^{n \times n}$ 是一个对角阵, 其定义为:

$$\mathbf{S} = \begin{bmatrix} t_1(1-t_1) & 0 & \dots & 0 \\ 0 & t_2(1-t_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & t_n(1-t_n) \end{bmatrix} \quad (6-23)$$

注意, 此处 $0 < t_i < 1$, 因此对角元素满足 $t_i(1-t_i) > 0$ 。因此, 矩阵 \mathbf{S} 是正定矩阵。根据优化方面的知识, 这里的交叉熵损失函数是严格凸函数。我们知道这也意味着它有唯一的全局最优解, 并且局部最优解就是全局最优解。

下面的问题就是如何求出使得交叉熵损失函数最小的 \mathbf{w} 。从 \mathbf{g} 的计算公式中可以看出, 它跟当前的“错误项” $t_i - y_i$ 直接相关。与我们在线性回归中的处理办法类似, 一种朴素的想法是令 $\mathbf{g} = 0$ 来直接求解 \mathbf{w} 。在线性回归中, 损失函数是关于所求参数 \mathbf{w} 的二次函数, 因此可以直接推导出解析解。但注意到 $t_i = \text{sig}(\mathbf{w}^T \mathbf{x}_i)$, 换言之, 因为非线性的 sigmoid 函数的引入, 所以 \mathbf{w} 是以非线性的形式出现在导数 \mathbf{g} 中的, 使得我们无法得到关于 \mathbf{w} 的解析解。因此, 在这种情况下, 我们一般使用数值解法, 采用逐步逼近的方法来求出 \mathbf{w} 的最优解。

6.3.3 牛顿法

在求解逻辑回归时, 我们一般使用牛顿法来求得最优解。在该算法中, 每一步我们都假设可以用一个二次函数来近似我们要最小化的目标函数, 从而逐步接近最优解。

首先回忆一下泰勒展式。在泰勒展式中, 对于函数 $f(w)$, 当 w 在 a 附近时可以使用如下展式来逼近 $f(w)$:

$$f(w) = f(a) + \frac{f'(a)}{1!}(w-a) + \frac{f''(a)}{2!}(w-a)^2 + \frac{f'''(a)}{3!}(w-a)^3 + \dots \quad (6-24)$$

在牛顿法中, 我们使用逐步逼近的方法来求解参数 w 。这里我们用下标 k 标识在第 k 步的诸变量值。例如, 用 w_k 表示第 k 步的 w 值。在第 k 步, 我们有当前估计 w_k , 就可以将 $f(w)$ 用其在 w_k 处的二阶泰勒展式来近似:

$$f(w) \approx f_{\text{quad}}(w) = f(w_k) + \frac{f'(w_k)}{1!}(w-w_k) + \frac{f''(w_k)}{2!}(w-w_k)^2 \quad (6-25)$$

上面的公式仅仅适用于一维的情况。在这里我们要优化的参数 $w \in \mathbb{R}^d$, 上面的二阶泰勒展式可以写为:

$$f(w) \approx f_{\text{quad}}(w) = f(w_k) + g_k^T(w-w_k) + \frac{1}{2}(w-w_k)^T H_k(w-w_k) \quad (6-26)$$

根据泰勒展式的原理, $f_{\text{quad}}(w)$ 在 w 接近 w_k 的时候是比较准确的。这里我们的目标是从 w_k 推导出计算 w_{k+1} 的方法。

这里为了简化推导过程, 我们将 $f_{\text{quad}}(w)$ 简写为:

$$f_{\text{quad}}(w) = \frac{1}{2}w^T H_k w + c^T w + d \quad (6-27)$$

这里 c 、 d 分别定义为 $c = g_k - H_k w_k$ 和 $d = f(w_k) - g_k^T w_k + \frac{1}{2}w_k^T H_k w_k$ 。

由于 $f_{\text{quad}}(w)$ 是关于 w 的二次函数, 通过求导并将其置为 0, 我们可以获得使得 $f_{\text{quad}}(w)$ 最小的最优解,

$$\begin{aligned} \frac{d}{dw} f_{\text{quad}}(w) &= H_k w + c = 0 \\ \Rightarrow \hat{w} &= -H_k^{-1}c = -H_k^{-1}(g_k - H_k w_k) = w_k - H_k^{-1}g_k \end{aligned} \quad (6-28)$$

因此, 在牛顿法中, 我们的更新公式是:

$$w_{k+1} = w_k - H_k^{-1}g_k$$

该更新公式是牛顿法迭代求解的核心。注意, 在该更新公式中, 我们要计算二阶导数 $H_k \in \mathbb{R}^{d \times d}$ 的逆矩阵。当数据的维数 d 很大时, 计算复杂度很大。在实际使用中, 牛顿法一般收敛较快, 只需要较少的迭代步数就可以达到较高的精度。

在前面的推导过程中, 我们直接利用了一阶导数 g_k 和二阶导数 H_k 。下面我们进一步简化逻辑回归中牛顿法的更新公式:

$$\begin{aligned}
\mathbf{w}_{k+1} &= \mathbf{w}_k - \mathbf{H}_k^{-1} \mathbf{g}_k \\
&= \mathbf{w}_k + \left(\mathbf{X}^T \mathbf{S}_k \mathbf{X} \right)^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{t}_k) \\
&= \left(\mathbf{X}^T \mathbf{S}_k \mathbf{X} \right)^{-1} \mathbf{X}^T (\mathbf{S}_k \mathbf{X} \mathbf{w}_k + \mathbf{y} - \mathbf{t}_k) \\
&= \left(\mathbf{X}^T \mathbf{S}_k \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{S}_k \mathbf{z}_k
\end{aligned} \tag{6-29}$$

这里 \mathbf{z}_k 定义为:

$$\mathbf{z}_k = \mathbf{X} \mathbf{w}_k + \mathbf{S}_k^{-1} (\mathbf{y} - \mathbf{t}_k) \tag{6-30}$$

因此, 每一步实质上是求解一个线性方程组:

$$\mathbf{X}^T \mathbf{S}_k \mathbf{X} \mathbf{w}_{k+1} = \mathbf{X}^T \mathbf{S}_k \mathbf{z}_k \tag{6-31}$$

回忆在最小二乘法中, 我们需要求解正规方程 (normal equation):

$$\mathbf{X}^T \mathbf{X} \mathbf{w}_{k+1} = \mathbf{X}^T \mathbf{y} \tag{6-32}$$

逻辑回归中的方程与正规方程非常相似。而且我们注意到矩阵 $\mathbf{S} \in \mathbb{R}^{d \times d}$ 是一个对角矩阵, 可以认为 $\sqrt{S_{ii}}$ 是数据 \mathbf{X} 中第 i 维的权重。因此, 求解逻辑回归的牛顿法也称为重新加权迭代最小二乘法 (iterative reweighted least squares)。

6.3.4 正则化项的引入

与回归问题中的最小二乘法类似, 也可以在逻辑回归中引入正则化项以控制模型的复杂度。根据前面讨论的交叉熵损失函数, 首先可以引入 L_2 范数:

$$E_2(\mathbf{w}) = -\sum_{i=1}^n (y_i \ln t_i + (1 - y_i) \ln(1 - t_i)) + \lambda_2 \|\mathbf{w}\|_2^2 \tag{6-33}$$

这里 $\lambda_2 \geq 0$ 是控制范数权重的系数。这里前半部使用交叉熵损失函数来度量模型 f 在训练数据上的表现, 后半部使用 L_2 范数来控制模型的复杂度。引入 L_2 范数后, 我们只需相应地更新一阶导数和二阶导数的计算公式, 仍然可以使用牛顿法求解。

与 Lasso 类似, 我们也可以引入 L_1 范数

$$E_1(\mathbf{w}) = -\sum_{i=1}^n (y_i \ln t_i + (1 - y_i) \ln(1 - t_i)) + \lambda_1 \|\mathbf{w}\|_1 \tag{6-34}$$

这里 $\lambda_1 \geq 0$ 是控制范数 $\|\mathbf{w}\|_1$ 权重的系数。因为 $\|\mathbf{w}\|_1$ 在 $\mathbf{w} = 0$ 点处连续但不可导, 所以在求解上更加复杂, 本书不作介绍。

与 Elastic Net 类似, 我们可以同时引入 L_1 范数和 L_2 范数:

$$E_{12}(\mathbf{w}) = -\sum_{i=1}^n (y_i \ln t_i + (1 - y_i) \ln(1 - t_i)) + \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2 \tag{6-35}$$

6.3.5 实际使用

本节我们使用 `glmnet` 包来讲解如何在 R 中应用逻辑回归，包括使用不同的 L_1 范数和 L_2 范数的权重。R 中的 `glm2` 包也实现了逻辑回归，但在本书中我们着重讨论 `glmnet` 包。本节完全的 R 代码在文件 `logistic_regression_example.R` 中。

在回归算法中，我们已经讨论过使用 `glmnet` 来求解 Lasso 了。在使用 `glmnet` 包中的 `glmnet` 函数求解逻辑回归时，主要有如下参数需要考虑：

- 参数 `family`，在回归中 `family` 应设置为 `'gaussian'`，在两类分类问题中应设置为 `'binomial'`；
- 参数 `lambda`；
- 参数 `alpha`。

回顾回归问题的讨论，我们知道参数 `lambda` 和 `alpha` 共同决定了 L_1 范数和 L_2 范数的权重。具体来说， L_1 范数的权重是 `lambda*alpha`， L_2 范数的权重是 `lambda*(1-alpha)/2`。此外还要注意，`glmnet` 的输入数据必须显式表示为矩阵的形式，而不能使用 R 中更常用的表示数据的数据框的形式。如果数据不是矩阵的形式，我们需要使用函数 `as.matrix()` 将输入数据转换为矩阵的形式。由于输入数据必须是矩阵的形式，因此，如果数据中有分类变量，我们也必须按照 5.5.2 节中转换分类变量的方法对其进行转换。

在得到 `glmnet` 函数训练好的模型后，可以调用 `predict`（其实是 `predict.glmnet`）函数来预测新的数据对应的类别或者属于某一类的概率。`predict.glmnet` 中的参数 `newx` 表示新的数据集。该函数中最重要的参数是 `type`。对于该参数使用不同的值可以得到不同的预测结果，常用的选择项包括：

- `'link'`：在分类和回归问题中都返回 $\mathbf{w}^T \mathbf{x}$ 。注意，在回归问题中， $\mathbf{w}^T \mathbf{x}$ 为拟合值。
- `'response'`：在分类问题中给出 $\text{sig}(\mathbf{w}^T \mathbf{x})$ ，即为测试数据属于正类的概率；在回归问题中为拟合值 $\mathbf{w}^T \mathbf{x}$ 。
- `'class'`：在分类问题中给出最后的分类结果，即对每个测试样本预测的类别。

在 R 中，类标列一般是因子类型，其中“正类”指的是训练数据集中类标列因子类别中的第二项。假设 `y_train` 是训练数据集对应的类别且类型为因子，则“正类”对应于 `levels(y_train)[2]`。

下面具体讲解文件 `logistic_regression_example.R` 中的 R 代码。首先检查如下 3 个包是否已经安装：

- `glmnet`
- `mlbench`
- `pROC`

这里使用包 `mlbench` 中的 Sonar 数据，使用 `pROC` 包中的 `auc` 函数计算 AUC 指标（AUC 是衡量算法分类性能的指标，详见 6.7 节）。如果没有安装的话，那么首先安装这 3 个包及它

们所依赖的包。

在正式使用 `glmnet` 构建逻辑回归模型之前, 使用如下代码来导入数据, 检查导入的数据框中每列的数据类型, 并打印数据框的前 10 行和最后 10 行。注意, 数据框 `D` 的最后一列 (列名为 `Class`) 保存了每行对应的样本的类别信息。

```
data(Sonar)
D <- Sonar
# show the type for each column
str(D)
# Check the first 10 rows and save them in LaTeX format
D_h10 <- head(D, 10)
print('the first 10 rows are:')
print(D_h10)
D_t10 <- tail(D, 10)
print('the last 10 rows are:')
print(D_t10)
```

之后将数据划分为训练集和测试集。使用训练集训练逻辑回归模型, 并利用所得的模型计算其在测试集上的分类结果, 最后计算分类的准确率和 AUC。注意, 由于 `glmnet` 的输入数据 X 必须是矩阵的形式, 因此我们使用 `as.matrix()` 函数将输入数据从数据框的格式显式转换为矩阵的格式。这里还使用了 `set.seed()` 函数设置随机数生成器的种子以使得每次运行可以得到相同的结果。这里 `sample(n,k)` 函数从 $1 \sim n$ 的整数中随机选取 k 个不重复的整数。`X_train$Class <- NULL` 表示删除 `X_train` 中的 `Class` 列。

```
set.seed(13)
train_ratio <- 0.8
n_total <- nrow(D)
n_train <- round(train_ratio * n_total)
n_test <- n_total - n_train
list_train <- sample(n_total, n_train)
y_train <- D[list_train, 'Class']
y_test <- D[-list_train, 'Class']
X_train <- D[list_train,]
X_train$Class <- NULL
X_train <- as.matrix(X_train)
X_test <- D[-list_train,]
X_test$Class <- NULL
X_test <- as.matrix(X_test)
```

接下来使用训练数据 `X_train` 和 `y_train`, 由易到难地建立多个不同的逻辑回归模型。

在第 1 个逻辑回归模型中, 我们将 `lambda` 参数设为 0, 这样在训练模型时没有使用任何的正则化项。所得的模型记为 `M1`, 之后使用 `predict` 函数得到测试集的预测结果, 其中 `y_test_predict1` 是直接的分类结果, `y_test_prob1` 保存了测试集中每个样本属于 'R' 类 (该数据有两类, 分别为 'M' 和 'R') 的概率, 而 `y_test_raw1` 保存了 $w^T x$ 的值。也就是说, $y_test_prob1 = 1 / (1 + \exp(-y_test_raw1))$ 。利用 `y_test_predict1`, 可以计算该模型在测

试集上的准确率；利用 `y_test_prob1`，可以计算 AUC。这里简要介绍一下 `auc` 函数的使用方法。该函数的第一个参数是测试集的真实类别，第二个参数是预测的概率值（其实使用 `y_test_raw1` 也可以，只要能够将测试集中的样本属于正类的概率从高到低进行排序即可，不一定是概率值）。注意，`y_test_prob1` 是一个矩阵类型的对象，要将其显式转换为向量对象。这里我们使用 `as.numeric(y_test_prob1)` 进行转换；由于 `y_test_prob1` 是一个只含有一列的矩阵，因此使用 `y_test_prob1[,1]` 也可以。

```
family <- 'binomial'
lambda <- 0
alpha <- 0
# train a logistic regression model
M1 <- glmnet(X_train, y_train, family = family, lambda = lambda, alpha = alpha)
# make the prediction on the test data set and compute the performance
y_test_predict1 <- predict(M1, X_test, type='class')
y_test_prob1 <- predict(M1, X_test, type='response')
y_test_raw1 <- predict(M1, X_test, type='link')

# Compute accuracy and AUC
y_test_numeric <- ifelse(y_test==levels(y_test)[2], 1, 0)
accuracy1 <- sum(y_test==y_test_predict1)/n_test
auc1 <- auc(y_test_numeric, as.numeric(y_test_prob1))
msg <- paste('accuracy = ', accuracy1)
print(msg)
msg <- paste('auc = ', auc1)
print(msg)
```

对应的输出如下：

```
[1] "accuracy = 0.761904761904762"
[1] "auc = 0.776887871853547"
```

在第 2 个逻辑回归模型中，我们将 `alpha` 设为 1，这样就仅使用 L_1 范数作为正则化项；同时我们将 `lambda` 的值设为 0.05。在得到模型 `M2` 后，可以利用 `M2$beta` 查看所得模型的 w 的具体数值。对应的 R 代码如下：

```
family <- 'binomial'
lambda <- 0.05
alpha <- 1
M2 <- glmnet(X_train, y_train, family = family, lambda = lambda, alpha = alpha)
# make the prediction on the test data set and compute the performance
y_test_predict2 <- predict(M2, X_test, type='class')
y_test_prob2 <- predict(M2, X_test, type='response')
accuracy2 <- sum(y_test==y_test_predict2)/n_test
auc2 <- auc(y_test, as.numeric(y_test_prob2))
msg <- paste('accuracy = ', accuracy2)
print(msg)
msg <- paste('auc = ', auc2)
```

```
print(msg)
print('the coefficients of w are')
print(M2$beta)
```

M2\$beta 对应的输出为一个稀疏矩阵，可以看出很多分量为 0。

```
[1] "the coefficients of w are"
60 x 1 sparse Matrix of class "dgCMatrix"
```

```
      s0
```

```
V1 -1.17769904
```

```
V2 .
```

```
V3 .
```

```
V4 -1.41760932
```

```
V5 .
```

```
V6 .
```

```
V7 .
```

```
V8 .
```

```
V9 .
```

```
V10 .
```

```
V11 -2.76170230
```

```
V12 -1.15366714
```

```
V13 .
```

```
V14 .
```

```
V15 .
```

```
V16 0.07396110
```

```
V17 .
```

```
V18 .
```

```
V19 .
```

```
V20 -0.06880197
```

```
V21 -0.40401291
```

```
V22 .
```

```
V23 -0.73455379
```

```
V24 .
```

```
V25 .
```

```
V26 .
```

```
V27 .
```

```
V28 -0.11135632
```

```
V29 .
```

```
V30 .
```

```
V31 .
```

```
V32 .
```

```
V33 .
```

```
V34 .
```

```
V35 .
```

```
V36 1.43526484
```

```
V37 .
```

```
V38 .
```

```
V39 .
```

```
V40 .
```

```

V41 .
V42 .
V43 .
V44 .
V45 -2.56348735
V46 .
V47 .
V48 .
V49 -7.58532417
V50 .
V51 -10.02455351
V52 -23.31103698
V53 .
V54 .
V55 .
V56 .
V57 .
V58 .
V59 .
V60 .

```

在第3个逻辑回归模型中，我们只考虑 L_2 范数作为正则化项，其代码如下：

```

family <- 'binomial'
lambda <- 0.05
alpha <- 0
M3 <- glmnet(X_train, y_train, family = family, lambda = lambda, alpha = alpha)

```

在第4个逻辑回归模型中，我们同时考虑了 L_1 范数和 L_2 范数，其代码如下：

```

family <- 'binomial'
lambda <- 0.05
alpha <- 0.3
M4 <- glmnet(X_train, y_train, family = family, lambda = lambda, alpha = alpha)

```

其中 L_1 范数对应的权重是 $\text{lambda} \times \text{alpha} = 0.015$ ， L_2 范数的权重是 $\text{lambda} \times (1 - \text{alpha}) / 2 = 0.0175$ 。

在实际中，由于 lambda 和 alpha 是逻辑回归中最重要的两个参数，通常需要实验多个不同的值，得到多个不同的模型并从中选出最优的模型。在使用 `glmnet` 时，可以输入多个 lambda 值（但 alpha 的值只允许一个），让 `glmnet` 同时训练多个不同的模型。由于 `glmnet` 中算法实现的缘故，同时输入多个 lambda 的计算复杂度显著低于多次调用 `glmnet` 处理单个 lambda 值的计算复杂度。在具体使用 `glmnet` 时，我们通常将 `lambda_list` 中的 lambda 值从大到小排列好（事实上，即使不排好，`glmnet` 也会在内部按照从大到小的顺序排好，但我们建议还是先排好，这样可以减少使用中的错误）。

```

lambda_list <- seq(0.05, 0, by=-0.005)
alpha <- 0

```

```

lambda_num <- length(lambda_list)
accuracy_train_list <- rep(0, lambda_num)
accuracy_test_list <- rep(0, lambda_num)
family <- 'binomial'
M5_list <- glmnet(X_train, y_train, family = family, lambda=lambda_list, alpha=alpha)
y_test_predict_matrix <- predict(M5_list, X_test, type='class')
y_train_predict_matrix <- predict(M5_list, X_train, type='class')
# We compute accuracy for these models on training and test data sets
for (i in 1:lambda_num) {
  accuracy_train_list[i] <- sum(y_train==y_train_predict_matrix[,i])/n_train
  accuracy_test_list[i] <- sum(y_test==y_test_predict_matrix[,i])/n_test
}

```

这里 `lambda_list` 包含 11 个不同的 `lambda` 值；与此对应，`M5_list` 包含了 11 个不同的模型，且 `M5_list$lambda` 记录了每个模型对应的 `lambda` 值（严格按从大到小的顺序排列）。因此，如果输入的 `lambda_list` 未排好的话，就要从 `M5_list$lambda` 而不是 `lambda_list` 中找到每个模型对应的 `lambda`。使用 `predict` 函数计算测试集中数据的预测值时，我们调用了所有的 11 个模型。在这个例子中，`y_test_predict_matrix` 和 `y_train_predict_matrix` 都是包含 11 列的矩阵，每列对应一个模型。

由于 `glmnet` 内部能够同时处理多个不同的 `lambda` 值，因此，使用下面的代码，`glmnet` 可以自动测试多个不同的 `lambda` 值，并返回逻辑回归模型中参数 w 随着 `lambda` 的值变化的情况。

```

M6 <- glmnet(X_train, y_train, family = family, alpha=1)
plot(M6)
print(M6)
coef6 <- coef(M6, s=0.01)
y_test_predict6 <- predict(M6, X_test, s=c(0.005, 0.01), type='class')

```

其中 `plot(M6)` 输出解 w 随着 `lambda` 的变化情况，如图 6-10 所示。图 6-10 中每条曲线对应一个变量（这里有 60 个变量）， x 轴是 $\|w\|_1$ ， y 轴对应 w 中每个分量的大小。可以看出，当 $\|w\|_1$ 变小（此时 `lambda` 值变大）时， w 中越来越多的分量变为 0。使用 `print(M6)` 可以得到不同的 `lambda` 值及 w 中非零分量的数目。此外，使用 `glmnet` 包中的 `coef` 函数，可以得到 `lambda` 取不同值时 w 的具体值（注意，在 `coef` 函数中，使用参数 `s` 表示 `lambda` 的值）。在上面的程序中，`coef6` 保存了当 `lambda` 为 0.01 时的 w 的具体值。对于 `M6`，在使用 `predict` 函数时，可以指定参数 `s` 为单个值或者多个值，`predict` 函数可以分别计算当 `s` 取不同值时对应模型的预测值。在上面的代码中，`s` 为 0.005 和 0.01，所得的 `y_test_predict6` 是一个包含两列的矩阵，每列对应一个 `s` 值。

`glmnet` 包也支持交叉检验以选出最优的模型。在 6.6 节会详细讨论交叉检验，这里简单介绍一下 `glmnet` 包中的 `cv.glmnet` 函数。在 `cv.glmnet` 函数中，使用 `nfolds` 参数指定交叉检验的重数，利用 `type.measure` 参数指定要优化的性能指标。在分类问题中，

type.measure 的常用值包括:

- 'class': 表示要优化分类的准确率;
- 'auc': 表示要优化分类的 AUC。

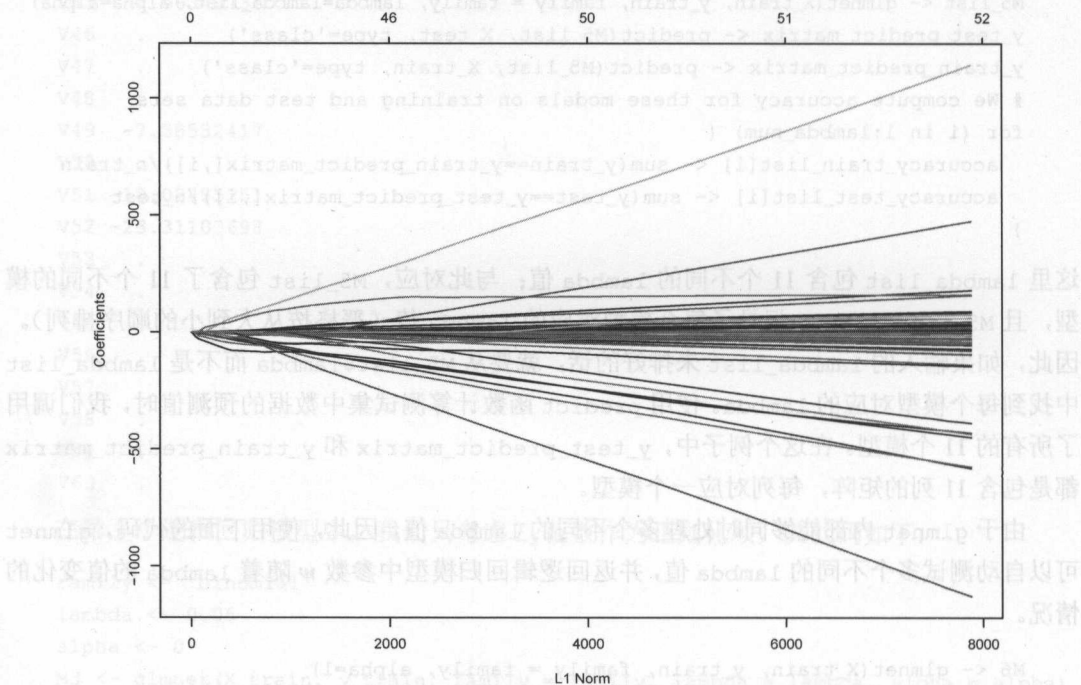


图 6-10 只使用 L_1 范数时 w 与 L_1 范数的关系

在下面的代码中,使用 5 重交叉检验以选出最优的模型。使用 `plot(M_cv)`,可以得到检验集中错误率随 `lambda` 的变化情况,如图 6-11 所示。其中第一条竖直直线对应使得检验值中准确率最高(即错误率最低)的 `lambda` 值。也可以直接使用 `M_cv$lambda.min` 得到最优的 `lambda` 值。类似地,可以使用 `coef` 函数得到 `lambda.min` 对应的 w 值。

```
M_cv = cv.glmnet(X_train, y_train, family = family, type.measure = 'class', nfolds=5)
plot(M_cv)
# the optimal lambda
M_cv$lambda.min
# get the coefficient corresponding the optimal lambda
coef7 <- coef(M_cv, s = "lambda.min")
# make prediction using the optimal lambda value
y_test_predict7 <- predict(M_cv, X_test, s="lambda.min", type='class')
```

在前面的 4 个逻辑回归模型中,指定单个的 `lambda` 和 `alpha` 值并得到对应的模型。在实际使用中,通常考虑一系列的 `lambda` 值,为每个 `lambda` 值训练一个逻辑回归模型,并从中选取最优的模型。因此,后面讲解的 `glmnet` 使用方法在实际中使用更多一些。

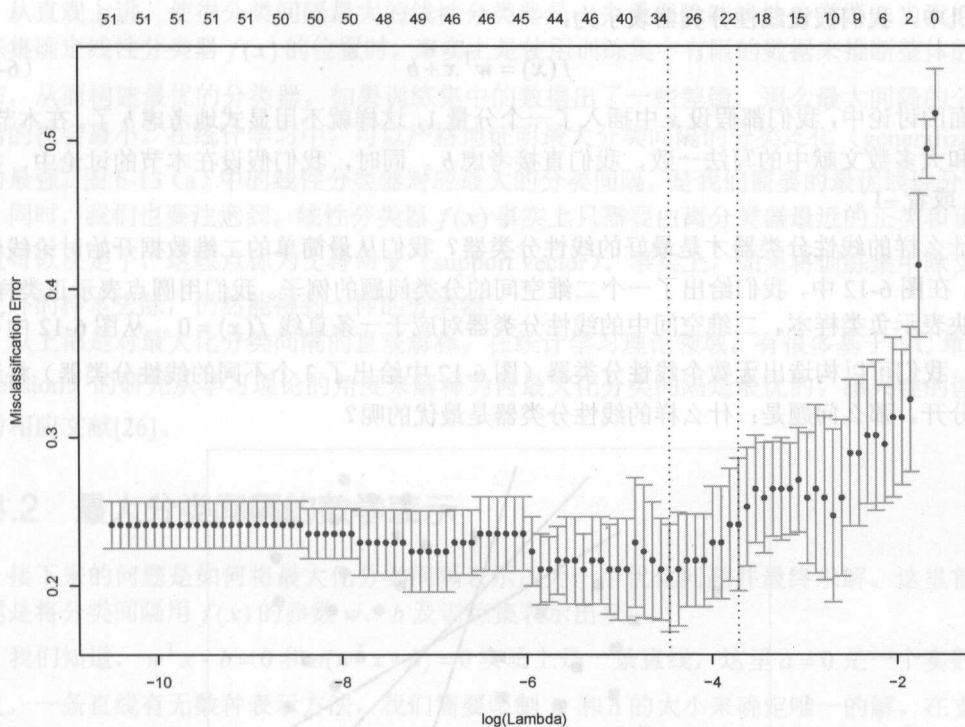


图 6-11 使用 `cv.glmnet` 时检验集上错误率随 λ 的变化情况

6.4 支持向量机

支持向量机 (support vector machine, SVM) 发端于统计学习理论, 具有非常坚实的统计学习理论支撑。支持向量机最初于 1992 年提出, 随后由于其在手写数字识别上的成功应用, 迅速在很多领域取得了广泛的应用。特别是 SVM 与核方法 (kernel method) 紧密联系起来, 成为机器学习前些年较热门的研究方向之一。本节首先介绍 SVM 的基本思想, 即最大化分类间隔。从最大化分类间隔出发, 我们讨论 SVM 在训练数据线性可分及线性不可分情况下的具体形式, 并介绍 Hinge 损失函数。然后, 简要介绍 SVM 中所得优化问题的对偶形式以求解 SVM; 在此基础上, 引入核 (kernel) 的概念并简要介绍常用的核函数。最后, 介绍 R 中的 `e1071` 包, 讲解如何使用该包来对实际数据进行分类。

6.4.1 基本思想: 最大化分类间隔

在前一节中我们已经讨论了线性分类器。线性分类器的核心思想是在特征空间中找出一个超平面来将正类样本和负类样本分开。在逻辑回归中, 分类的超平面由使得 $\text{sig}(\mathbf{w}^T \mathbf{x}) = 0.5$ 的点组成, 换言之, 就是由使得 $\mathbf{w}^T \mathbf{x} = 0$ 的点组成。支持向量机也属于线性分类器。在支持

向量机中，我们假设线性分类器表示为：

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b \quad (6-36)$$

在前面的讨论中，我们都假设 \mathbf{x} 中插入了一个分量 1，这样就不用显式地考虑 b 了。在本节中，为了和大多数文献中的写法一致，我们直接考虑 b 。同时，我们假设在本节的讨论中，类标 $y_i = 1$ 或者 -1 。

什么样的线性分类器才是最好的线性分类器？我们从最简单的二维数据开始讨论线性分类器。在图 6-12 中，我们给出了一个二维空间的分类问题的例子。我们用圆点表示正类样本，用方块表示负类样本。二维空间中的线性分类器对应于一条直线 $f(\mathbf{x}) = 0$ 。从图 6-12 中可以看出，我们可以构造出无数个线性分类器（图 6-12 中给出了 3 个不同的线性分类器）将这组数据分开。那么问题是：什么样的线性分类器是最优的呢？

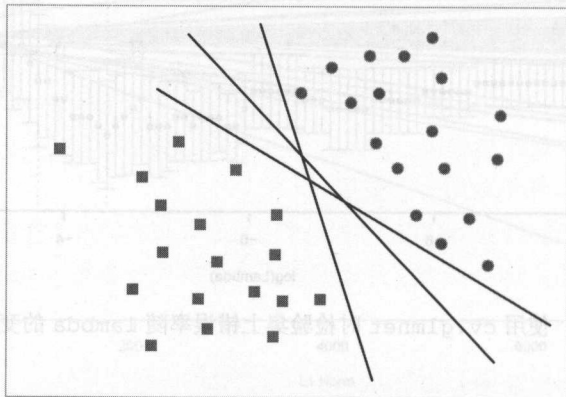


图 6-12 不同的线性分类器都可以将这组数据分开

对于某个给定的线性分类器，可以定义它所对应的分类间隔（classification margin）。这里分类间隔的定义为从 $f(\mathbf{x}) = 0$ 出发，离 $f(\mathbf{x})$ 最近的正类样本到 $f(\mathbf{x})$ 的距离加上离 $f(\mathbf{x})$ 最近的负类样本到 $f(\mathbf{x})$ 的距离。图 6-13 中显示了两个线性分类器及它们对应的分类间隔。

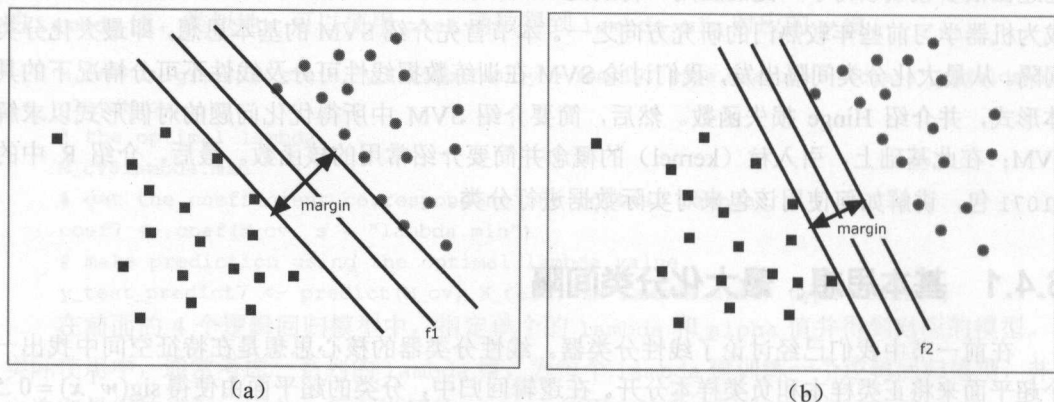


图 6-13 两个不同的线性分类器 f_1 (a) 和 f_2 (b) 及它们对应的分类间隔

从直观上讲,使得分类间隔最大的线性分类器是一个最保险的线性分类器。当我们使用训练集确定线性分类器 $f(\mathbf{x})$ 的位置时,事实上是使用训练集中有限的数据来推断整体的样本分布,从而构建最优的分类器。如果训练集中的数据出了一些差错,那么最大间隔的分类器犯错的机会最小。在统计学习中,可以严格地证明最大分类间隔的分类泛化 (generalization) 能力最强。图 6-13 (a) 中的线性分类器对应最大的分类间隔,是我们需要的最优线性分类器。

同时,我们也要注意,线性分类器 $f(\mathbf{x})$ 事实上只需要由离分类器最近的正类和负类样本就可以决定了,这些点称为支持向量 (support vector)。事实上,如果将训练集中除支持向量之外的样本去除,仍然能得到一样的分类器。

以上都是对最大化分类间隔的直观解释。在统计学习理论领域,有很多基于 VC 维 (VC dimension) 的研究从学习理论的角度来解释为何最大化分类间隔是最优的。感兴趣的读者可参考相应文献[26]。

6.4.2 最大分类间隔的数学表示

接下来的问题是如何将最大化分类间隔表示成为一个优化问题并最终求解。这里首要的问题是将分类间隔用 $f(\mathbf{x})$ 的参数 \mathbf{w} 、 b 及训练集表示出来。

我们知道, $\mathbf{w}^T \mathbf{x} + b = 0$ 和 $a(\mathbf{w}^T \mathbf{x} + b) = 0$ 实质上是一条直线,这里 $a \neq 0$ 是一个实数。换言之,一条直线有无数种表示方法,我们需要限制 \mathbf{w} 和 b 的大小来确定唯一的解。在支持向量机中,一般规定分类间隔由直线 $\mathbf{w}^T \mathbf{x} + b = 1$ 和 $\mathbf{w}^T \mathbf{x} + b = -1$ 确定,其中直线 $\mathbf{w}^T \mathbf{x} + b = 1$ 经过最靠近分类边界的正类样本,直线 $\mathbf{w}^T \mathbf{x} + b = -1$ 经过最靠近分类边界的负类样本,而分类边界由直线 $\mathbf{w}^T \mathbf{x} + b = 0$ 确定,如图 6-14 所示。

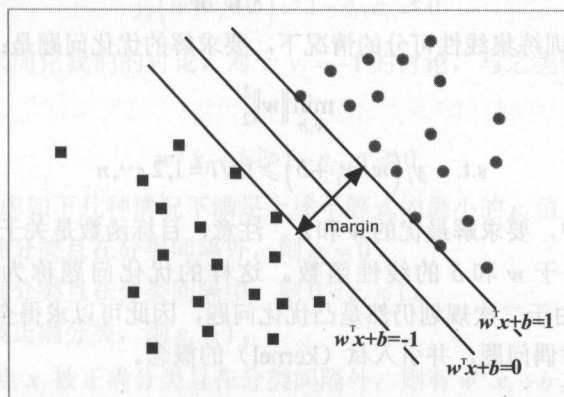


图 6-14 分类中的 3 条直线: $\mathbf{w}^T \mathbf{x} + b = -1, 0, 1$

这样,对于正类样本 (即 $y_i = 1$), 我们要求:

$$\mathbf{w}^T \mathbf{x}_i + b \geq 1 \quad (6-37)$$

对于负类样本 (即 $y_i = -1$), 我们要求:

$$\mathbf{w}^T \mathbf{x}_i + b \leq -1 \quad (6-38)$$

综合起来可以将约束条件写成:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad (6-39)$$

在支持向量机中,我们的目标是最大化分类间隔,因此,要推导出给定训练集和分类器 $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ 的情况下分类间隔的数学表示。利用解析几何的知识,我们知道给定一个点 \mathbf{x} , 它到直线 $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ 的距离为:

$$d(\mathbf{x}) = \frac{|\mathbf{w}^T \mathbf{x} + b|}{\sqrt{\mathbf{w}^T \mathbf{w}}} = \frac{|\mathbf{w}^T \mathbf{x} + b|}{\sqrt{\sum_{i=1}^d w_i^2}} \quad (6-40)$$

这里 $\mathbf{w} = [w_1, w_2, \dots, w_d]^T \in \mathbb{R}^d$ 。

根据之前的假设,分类间隔就是直线 $\mathbf{w}^T \mathbf{x} + (b-1) = 0$ 和直线 $\mathbf{w}^T \mathbf{x} + (b+1) = 0$ 之间的距离。在直线 $\mathbf{w}^T \mathbf{x} + (b+1) = 0$ 上任取一点 \mathbf{x}_0 , 则该点满足:

$$\mathbf{w}^T \mathbf{x}_0 + b = -1 \quad (6-41)$$

这样我们可以直接利用前述的点到直线的距离公式,得到 \mathbf{x}_0 到 $\mathbf{w}^T \mathbf{x} + b = 1$ 的距离为:

$$d = \frac{|\mathbf{w}^T \mathbf{x}_0 + (b+1)|}{\sqrt{\mathbf{w}^T \mathbf{w}}} = \frac{2}{\sqrt{\mathbf{w}^T \mathbf{w}}} \quad (6-42)$$

根据分类间隔的计算公式,我们需要最大化 $\frac{2}{\sqrt{\mathbf{w}^T \mathbf{w}}}$, 等价于最小化 $\sqrt{\mathbf{w}^T \mathbf{w}}$, 即等价于最小化 $\|\mathbf{w}\|_2^2$ 。因此,在训练集线性可分的情况下,要求解的优化问题是:

$$\begin{aligned} \min_{\mathbf{w}, b} & \|\mathbf{w}\|_2^2 \\ \text{s.t. } & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \forall i = 1, 2, \dots, n \end{aligned} \quad (6-43)$$

在这个优化问题中,要求解最优的 \mathbf{w} 和 b 。注意,目标函数是关于 \mathbf{w} 的二次函数,而所有的约束条件都是关于 \mathbf{w} 和 b 的线性函数。这样的优化问题称为二次规划 (quadratic programming, QP)。由于二次规划仍然是凸优化问题,因此可以求得全局最优解。我们在下面讨论该优化问题的对偶问题,并引入核 (kernel) 的概念。

6.4.3 如何处理线性不可分的数据

在实际应用中,很多时候数据是线性不可分的。换言之,不存在一个线性分类器 $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ 能把所有的正类样本和负类样本分开,如图 6-15 所示。在这种情况下,理想的线性分类器应该既能够最大化分类间隔,又能减少错误分类的样本数目。

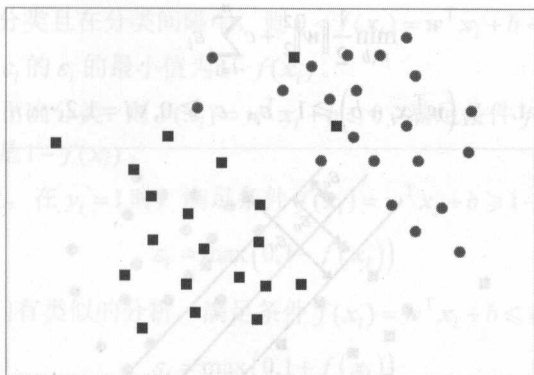


图 6-15 线性不可分的二维数据

既然不能将训练集中的所有样本都正确分类，那么一个很自然的想法是尽量多地正确分类训练集中的样本。换言之，就是最大化分类准确率。但在实际的分类算法中，很少有算法去直接最大化准确率（也即最小化错误分类的样本数），原因包括：（1）不好用简单的数学公式表示分类准确率；（2）就算表示出来，一般也不易求解所得的优化问题。

因此，很多分类算法都不直接处理错误分类的样本数，而用一些在优化问题中容易处理的“近似项”来逼近它，从而求得最优解。具体来说，在支持向量机中，当样本不能使用线性分类器直接分类时，对于训练集中的每个样本引入松弛变量 ε_i ，并且用松弛变量 ε_i 的和来近似错误分类的样本数，从而求解相应的优化问题。

严格地讲，对于每个样本，引入松弛变量 ε_i ，并要求下面的约束条件成立：

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \varepsilon_i, \varepsilon_i \geq 0 \quad (6-44)$$

这里假设 $y_i = 1$ 来简化我们的讨论，对于 $y_i = -1$ 的讨论，与之类似。由上式，约束条件变为：

$$\mathbf{w}^T \mathbf{x}_i + b \geq 1 - \varepsilon_i, \varepsilon_i \geq 0 \quad (6-45)$$

对于样本 \mathbf{x}_i ，考虑如下几种情况下满足上述不等式的最小的 ε_i 值：

- 如果 \mathbf{x}_i 被正确分类且在分类间隔外，则 $\varepsilon_i = 0$ 。
- 如果 \mathbf{x}_i 被正确分类且在分类间隔中，则 $0 < \varepsilon_i < 1$ 。
- 如果 \mathbf{x}_i 没有被正确分类，则 $\varepsilon_i \geq 1$ 。

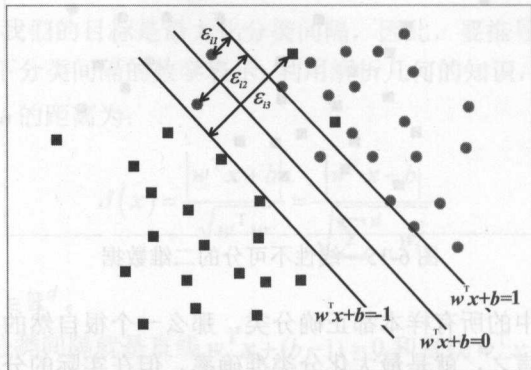
具体而言，如果点 \mathbf{x}_i 被正确分类且在分类间隔外，则有 $\mathbf{w}^T \mathbf{x}_i + b \geq 1$ ，此时可取 $\varepsilon_i = 0$ 。如果 \mathbf{x}_i 被正确分类且在分类间隔中，当 \mathbf{x}_i 是正类时，我们有 $0 < \mathbf{w}^T \mathbf{x}_i + b < 1$ ，此时可取 $0 < \varepsilon_i < 1$ 。如果正类样本 \mathbf{x}_i 被错误地分成负类，则 $\mathbf{w}^T \mathbf{x}_i + b \leq 0$ ，此时可取 $\varepsilon_i \geq 1$ 。综合考虑这 3 种情况， $\sum_{i=1}^n \varepsilon_i$ 是错误分类的样本数的上界，即 $\sum_{i=1}^n \varepsilon_i$ 始终大于或等于被错分的样本数。图 6-16 给出了 3 个点对应的 ε_i 。从图 6-16 中我们可以清楚地理解 ε_i 的几何意义。

严格地讲，在线性不可分的情况下我们最小化如下函数：

$$\min_{w,b} \frac{1}{2} \|w\|_2^2 + c \sum_{i=1}^n \varepsilon_i$$

(6-46)

$$\text{s.t. } y_i (w^T x_i + b) \geq 1 - \varepsilon_i, \quad \varepsilon_i \geq 0, \quad \forall i = 1, 2, \dots, n$$

图 6-16 数据不是线性可分时所引入的惩罚项 ε_i

这里的目标函数由两部分组成：第一部分对应于最大化分类间隔，第二部分对应于训练集中被错误分类的样本数的上界。注意，我们给第一部分使用了系数 $1/2$ 是为了便于下面的推导。第一部分和第二部分的权重由参数 c 来控制。一般在实际使用中，我们使用 6.6 节介绍的交叉检验来确定 c 的最优值。

6.4.4 Hinge 损失函数

在逻辑回归中我们已经讨论了交叉熵损失函数，本节讨论支持向量机中使用的 Hinge 损失函数。

在分类问题中，最直接的损失函数就是所谓的“0-1”损失函数，定义如下：

$$L_{01}(y, f) = \begin{cases} 1, & \text{如果 } y \neq f(x) \\ 0, & \text{如果 } y = f(x) \end{cases} \quad (6-47)$$

0-1 损失函数的定义直接明了：如果对于样本 x 分类正确，则损失函数为 0；否则为 1。那么，最小化 0-1 损失函数就是直接最小化被错分的样本数目。但 0-1 损失函数在实际中很少使用，因为它既不好表示，也不好求导，一般用在算法的性能评价中。实际上，在机器学习的很多算法中，我们都使用一个近似函数来逼近 0-1 损失函数作为要优化的对象。

在支持向量机中，我们使用 ε_i 的和作为 0-1 损失函数的上界，将所得的损失函数称为 Hinge 损失函数。我们再回到 $\sum_{i=1}^n \varepsilon_i$ 的分析。对于样本 x_i ，假设 $y_i = 1$ ，下面具体考虑 ε_i 与 $f(x_i) = w^T x_i + b$ 的关系。

- 如果 x_i 被正确分类且在分类间隔外，则 $f(x_i) = w^T x_i + b \geq 1$ 。满足条件 $f(x_i) = w^T x_i + b \geq 1 - \varepsilon_i$ 的 ε_i 的最小值为 $\varepsilon_i = 0$ 。

- 如果 x_i 被正确分类且在分类间隔中, 则 $0 < f(x_i) = w^T x_i + b < 1$ 。满足条件 $f(x_i) = w^T x_i + b \geq 1 - \varepsilon_i$ 的 ε_i 的最小值为 $1 - f(x_i)$ 。
- 如果 x_i 没有被正确分类, 则 $f(x_i) = w^T x_i + b \leq 0$ 。满足条件 $f(x_i) = w^T x_i + b \geq 1 - \varepsilon_i$ 的 ε_i 的最小值也是 $1 - f(x_i)$ 。

综合以上 3 种情况, 在 $y_i = 1$ 时, 满足条件 $f(x_i) = w^T x_i + b \geq 1 - \varepsilon_i$ 的 ε_i 的最小值为:

$$\varepsilon_i = \max(0, 1 - f(x_i)) \quad (6-48)$$

当 $y_i = -1$ 时, 我们有类似的分析, 满足条件 $f(x_i) = w^T x_i + b \leq -1 - \varepsilon_i$ 的 ε_i 的最小值为:

$$\varepsilon_i = \max(0, 1 + f(x_i)) \quad (6-49)$$

将以上两种情况综合起来, 满足条件 $y_i(w^T x_i + b) \geq 1 - \varepsilon_i$ 的 ε_i 的最小值为:

$$\varepsilon_i = \max(0, 1 - y_i f(x_i)) \quad (6-50)$$

根据上面的分析, 我们引入 Hinge 损失函数, 其严格定义如下:

$$h(y, f) = \max(0, 1 - yf(x)) \quad (6-51)$$

图 6-17 给出了 Hinge 函数和 0-1 函数的图像, 其中横坐标是 yf (这里我们仍然假定 y 的取值为 1 或者 -1), 纵坐标是对应的函数值。0-1 函数无论错误多大, 只要错了 (当 $yf \leq 0$ 时), 就给出惩罚 1; 只要分类正确 (当 $yf > 0$ 时), 则不给予惩罚。根据 Hinge 函数的图像, 我们可以看到 f 如果分类错误 (当 $yf \leq 0$ 时), 错误越大, Hinge 函数给予的惩罚越大; 而且就算是 f 分类正确, 如果 yf 的值小于 1, Hinge 函数仍然给予惩罚。在前面的讨论中, 我们指出 $\sum_{i=1}^n \varepsilon_i$ 是被错分的样本数, 即 0-1 损失函数的上界。从图 6-17 中也可以明确地看出, Hinge 函数的值一直在 0-1 损失函数之上。

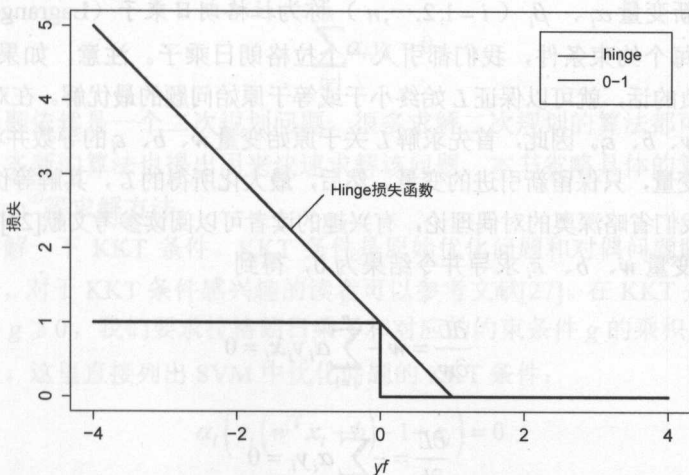


图 6-17 Hinge 损失函数

损失函数在机器学习中非常重要,它定义了模型 f 在训练集上对于训练数据的拟合好坏。除了损失函数,在机器学习中,我们一般在待优化的目标函数中添加正则化项来控制模型的复杂度,避免所得到的模型在训练集上过拟合。在支持向量机中,我们知道所对应的损失函数是 Hinge 函数,而正则化项实际对应于分类器的分类间隔。从直观的角度来看,分类间隔对应着模型的复杂度。分类间隔越大,模型的复杂度越低。

6.4.5 对偶问题

接下来我们讨论支持向量机中优化问题的对偶问题并引出核学习的基本概念。阅读这部分内容需要一些优化方面的知识,如果读者对于支持向量机的具体求解不感兴趣,那么可以直接跳至 6.4.7 节了解如何使用 R 中的软件包来使用支持向量机进行分类。

由于线性可分所对应的优化问题是线性不可分的特例,我们直接讨论线性不可分的情况。在线性不可分的情况下,需要求解如下优化问题:

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|_2^2 + c \sum_{i=1}^n \varepsilon_i \\ \text{s.t.} \quad & y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \varepsilon_i, \varepsilon_i \geq 0, \forall i = 1, 2, \dots, n \end{aligned} \quad (6-52)$$

这是一个二次规划问题。求解的难点在于对约束条件的处理。在优化理论中,利用拉格朗日函数可以处理约束条件。下面我们推导出该优化问题的对偶问题 (dual problem)。首先引入拉格朗日函数 L :

$$L = \frac{1}{2} \|\mathbf{w}\|_2^2 + c \sum_{i=1}^n \varepsilon_i - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \varepsilon_i) - \sum_{i=1}^n \beta_i \varepsilon_i \quad (6-53)$$

这里引入的新变量 α_i 、 β_i ($i = 1, 2, \dots, n$) 称为拉格朗日乘子 (Lagrange multiplier)。对于优化问题中的每个约束条件,我们都引入一个拉格朗日乘子。注意,如果限定所有的拉格朗日乘子都为非负的话,就可以保证 L 始终小于或等于原始问题的最优解。在对偶问题中,我们要消去原始变量 \mathbf{w} 、 b 、 ε_i 。因此,首先求解 L 关于原始变量 \mathbf{w} 、 b 、 ε_i 的导数并将其置为 0,从而在 L 中消去原始变量,只保留新引进的变量。然后,最大化所得的 L ,其解等价于原始优化问题的最优解。这里我们省略深奥的对偶理论,有兴趣的读者可以阅读参考文献[27]的第 5 章。

将 L 对原始变量 \mathbf{w} 、 b 、 ε_i 求导并令结果为 0, 得到

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = 0 \quad (6-54)$$

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^n \alpha_i y_i = 0 \quad (6-55)$$

$$\frac{\partial L}{\partial \varepsilon_i} = c - \alpha_i - \beta_i = 0 \quad (6-56)$$

因此, 有

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (6-57)$$

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad (6-58)$$

$$\alpha_i + \beta_i = c \quad (6-59)$$

将其代入回 L 中, 可得

$$\begin{aligned} L &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + c \sum_{i=1}^n \varepsilon_i - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \mathbf{w} - \sum_{i=1}^n \alpha_i y_i b + \sum_{i=1}^n \alpha_i - \sum_{i=1}^n \alpha_i \varepsilon_i - \sum_{i=1}^n \beta_i \varepsilon_i \\ &= \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n (c - \alpha_i - \beta_i) \varepsilon_i - \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i \quad (6-60) \\ &= -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \alpha_i \end{aligned}$$

在 Karush-Kuhn-Tucker (KKT) 条件中, 我们要求 $\alpha_i \geq 0, \beta_i \geq 0$ 。根据 $\alpha_i + \beta_i = c$ 可知 $0 \leq \alpha_i \leq c$ 。综上所述, 在对偶问题中, 我们要求解的优化问题是:

$$\begin{aligned} \max_{\alpha_1, \alpha_2, \dots, \alpha_n} \quad & -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq c \end{aligned} \quad (6-61)$$

$$\sum_{i=1}^n \alpha_i y_i = 0$$

这个对偶问题依然是一个二次规划问题。很多求解二次规划的算法都可以用来求解该问题。近年来, 很多新的算法也提出用来快速求解该问题。本书省略具体的算法, 感兴趣的读者可以参阅 libsvm^①等求解方法。

下面简单讲解一下 KKT 条件。KKT 条件是原始优化问题和对偶问题联系的纽带。我们这里只给出结论, 对于 KKT 条件感兴趣的读者可以参考文献[27]。在 KKT 条件中, 对于每条不等式约束条件 $g \geq 0$, 我们要求拉格朗日乘子和对应的约束条件 g 的乘积为 0, 且对应的拉格朗日乘子非负。这里直接列出 SVM 中优化问题的 KKT 条件:

$$\alpha_i \left(y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \varepsilon_i \right) = 0$$

① <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

$$\beta_i \varepsilon_i = 0$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \varepsilon_i \geq 0 \quad (6-62)$$

$$\varepsilon_i \geq 0$$

$$\alpha_i \geq 0$$

$$\beta_i \geq 0$$

从原始问题的解我们已知 $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$ 。如果点 \mathbf{x}_i 被正确分类且在分类间隔之外, 即

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) > 1 \quad (6-63)$$

则 $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \varepsilon_i > 0$ 。由条件 $\alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \varepsilon_i) = 0$ 可知 $\alpha_i = 0$; 换言之, 计算 \mathbf{w} 时可省略该点。而那些位于分类间隔之间的点及错分的点, 其对应的 $\alpha_i > 0$; 换言之, \mathbf{w} 只由那些使得 $\alpha_i > 0$ 的点决定, 我们把这样的点 \mathbf{x}_i 称为支持向量。

6.4.6 非线性支持向量机和核技巧

在前面的对偶问题中, 我们注意到, 给定训练集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, 在学习 \mathbf{w} 和 b 的时候, 对于 $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, 我们只用到了 $\mathbf{x}_i^T \mathbf{x}_j$ 。利用所得的分类器 $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ 去预测一个新样本 \mathbf{x} 时, 我们有:

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + b \\ &= \left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right)^T \mathbf{x} + b \\ &= \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b \end{aligned} \quad (6-64)$$

可以看出, 在预测的时候我们同样只需要计算 $\mathbf{x}_i^T \mathbf{x}$ 的值即可。因此, 在应用支持向量机时, 除了类标信息 y_i 外, 数据本身的信息只需要用到样本间的内积 $\mathbf{x}_i^T \mathbf{x}_j$ 。

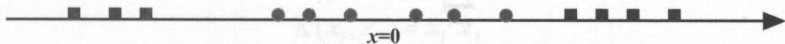
另一方面, 在前面所讨论的支持向量机中, 我们所求的是特征空间中的一个超平面。在实际中, 数据的分布可能很复杂, 这时仅仅使用一个超平面无法将正负类的样本分开。在核学习中, 我们将数据从原来的低维空间映射到高维甚至无限维空间中, 使得正负类的样本能够新的空间中容易分类。将数据映射到更高维空间的做法, 与机器学习中处理数据的常用方法相悖。特别是高维数据容易带有冗余信息, 引起维数灾难 (curse of dimensionality)。但在一些实际问题中, 我们的确需要将数据映射到更高维空间中。下面用一个简单的分类问题来说明映射到高维空间的必要性。

图 6-18 (a) 给出了一组无法使用线性分类器分类的一维数据。图 6-18 (a) 中圆点对应一类, 方块对应另一类。可以看出, 在一维空间中无论如何构建线性分类器, 都无法完美地

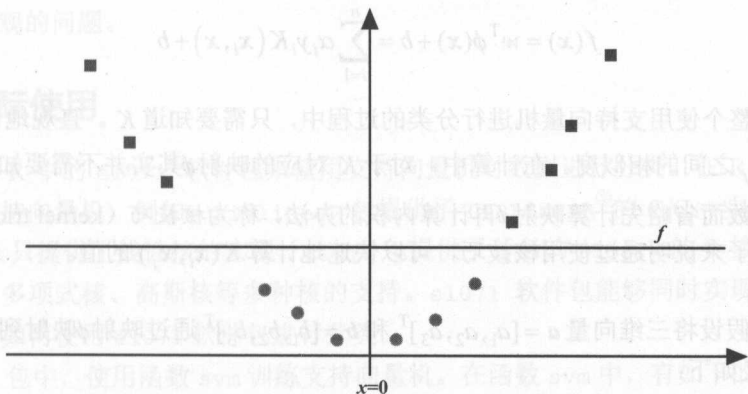
将两类分开。为了将两类样本点分开，使用如下映射将数据从一维空间映射到二维空间中：

$$\phi(x) = [x, x^2]^T \quad (6-65)$$

图 6-18 (b) 显示了映射到二维空间后数据的分布。可以看出，在二维空间中，可以轻易地使用一条直线将两类样本分开。



(a) 在一维空间中无法使用线性分类器分类的一组数据



(b) 将 (a) 中数据投影到二维空间后，数据线性可分

图 6-18 映射到高维空间的必要性示例

在这个例子中，我们显式地定义了映射 $\phi(x)$ 。而核学习的另一个优点是并不需要直接知道映射 $\phi(x)$ ，而只需要知道 $\phi(x_i)^T \phi(x_j)$ ，即 x_i, x_j 映射到高维空间后的内积即可。下面利用支持向量机中的对偶问题来说明这一点。使用映射 $\phi(x)$ 后，支持向量机求解的对偶问题是：

$$\begin{aligned} \max_{\alpha_1, \alpha_2, \dots, \alpha_n} \quad & -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) + \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq c \end{aligned} \quad (6-66)$$

$$\sum_{i=1}^n \alpha_i y_i = 0$$

因此，在求解支持向量机的过程中，我们真正要利用到的信息是：

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j) \quad (6-67)$$

这里 K 称为核函数 (kernel function)。这样对偶问题可以写为：

$$\begin{aligned}
 \max_{\alpha_1, \alpha_2, \dots, \alpha_n} & -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^n \alpha_i \\
 \text{s.t.} & 0 \leq \alpha_i \leq c \\
 & \sum_{i=1}^n \alpha_i y_i = 0
 \end{aligned} \quad (6-68)$$

当求得高维空间中的支持向量机模型后,可以利用所得的分类器 $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ 去预测一个新样本 $\phi(\mathbf{x})$:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i=1}^n \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \quad (6-69)$$

因此,在整个使用支持向量机进行分类的过程中,只需要知道 K 。直观地讲, $K(\mathbf{x}_i, \mathbf{x}_j)$ 反映了 \mathbf{x}_i 和 \mathbf{x}_j 之间的相似度。在计算中,对于 K 对应的映射 ϕ 其实并不需要知道。这种通过直接计算核函数而省略先计算映射 ϕ 再计算内积的办法,称为核技巧(kernel trick)。这里给出一个具体的例子来说明通过使用核技巧,可以快速地计算 $K(\mathbf{x}_i, \mathbf{x}_j)$ 的值。

例 6-4 假设将三维向量 $\mathbf{a} = [a_1, a_2, a_3]^T$ 和 $\mathbf{b} = [b_1, b_2, b_3]^T$ 通过映射 ϕ 映射到十维空间。其中映射 ϕ 的定义如下:

$$\begin{aligned}
 \phi(\mathbf{a}) &= [1, \sqrt{2}a_1, \sqrt{2}a_2, \sqrt{2}a_3, \sqrt{2}a_1a_2, \sqrt{2}a_1a_3, \sqrt{2}a_2a_3, a_1^2, a_2^2, a_3^2]^T, \\
 \phi(\mathbf{b}) &= [1, \sqrt{2}b_1, \sqrt{2}b_2, \sqrt{2}b_3, \sqrt{2}b_1b_2, \sqrt{2}b_1b_3, \sqrt{2}b_2b_3, b_1^2, b_2^2, b_3^2]^T
 \end{aligned} \quad (6-70)$$

这样,映射后计算 $\phi(\mathbf{a})$ 和 $\phi(\mathbf{b})$ 的内积可得:

$$\begin{aligned}
 \phi(\mathbf{a})^T \phi(\mathbf{b}) &= 1 + 2a_1b_1 + 2a_2b_2 + 2a_3b_3 + 2a_1a_2b_1b_2 + 2a_1a_3b_1b_3 + 2a_2a_3b_2b_3 + a_1^2b_1^2 + a_2^2b_2^2 + a_3^2b_3^2 \\
 &= (a_1b_1 + a_2b_2 + a_3b_3 + 1)^2 = (\mathbf{a}^T \mathbf{b} + 1)^2
 \end{aligned} \quad (6-71)$$

因此,在这个例子中,无须计算映射 $\phi(\mathbf{a})$ 和 $\phi(\mathbf{b})$,只需要计算 $(\mathbf{a}^T \mathbf{b} + 1)^2$ 就可以得到 $\phi(\mathbf{a})$ 和 $\phi(\mathbf{b})$ 的内积。

在实际使用支持向量机时,常用的核包括以下几个。

- 多项式核 (polynomial kernel):

$$K(\mathbf{x}_i, \mathbf{x}_j) = (R + \mathbf{x}_i^T \mathbf{x}_j)^d \quad (6-72)$$

这里 d 是正整数, $R \geq 0$ 为控制参数。

- 高斯核 (radial basis kernel):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right) \quad (6-73)$$

这里 $\sigma > 0$ 是控制参数。

- 线性核 (linear kernel): 当 $\phi(\mathbf{x}) = \mathbf{x}$ 时, 把所得的核称为线性核。

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \quad (6-74)$$

在实际中, 根据数据的具体特征, 用户可选择适当的核函数。通过使用核技巧, 能够将非线性引入到分类问题中, 同时计算复杂度也不高。但核的引入也可能导致过拟合的问题, 同时也有不直观的问题。

6.4.7 实际使用

这里利用 R 中的 e1071^① 软件包来使用支持向量机对数据进行分类。在 R 中, 有多种软件包实现了支持向量机, 例如, Liblinear^② 的 C/C++ 实现的 R 接口, 但 Liblinear 只提供了线性核的实现。e1071 包提供了著名的 LIBSVM 的 R 接口, 并提供了包括线性核、多项式核、高斯核等多种核的支持。e1071 软件包能够同时实现分类和回归, 本节主要介绍如何使用 e1071 软件包进行分类。

在 e1071 包中, 使用函数 svm 训练支持向量机。在函数 svm 中, 有如下的主要参数需要设置。

- 核的选定。在 e1071 中, 提供了线性核、高斯核、多项式核和 sigmoid 核。具体而言, 参数 kernel 可设为 'linear' (线性核)、'radial' (高斯核)、'polynomial' (多项式核) 和 'sigmoid' (sigmoid 核)。
- 对应核的参数。在高斯核中, 对应的参数是 gamma (这里 gamma 等于前面高斯核公式中的 $\frac{1}{2\sigma^2}$)。在多项式核中, 通过对应的参数 degree 设置多项式的次数, 通过 coef0 设定 R 。
- cost 值, 即支持向量机目标函数中正则化项的系数 c 。
- scaling 用来表示对各个特征是否进行正则化处理。默认设置为对所有变量都进行正则化处理。
- type 值用来指定是分类还是回归。在本节中全部使用默认设置 "C-classification"。

这里只介绍其中最重要的参数。关于 svm 的更多参数设置及 e1071 包中其他函数的使用, 可参见其手册^③。

① <https://cran.r-project.org/web/packages/e1071/index.html>

② <https://www.csie.ntu.edu.tw/~cjlin/liblinear/>

③ <http://cran.r-project.org/web/packages/e1071/e1071.pdf>

在样例程序 SVM_example1.R 中, 我们给出了 6 个例子。如设置一个线性核, 并将 `cost` 的值设为 1:

```
svm.linear.model1 <- svm(class ~., data = D_train, kernel = 'linear', cost = 1)
```

也可以改变 `cost` 的值为 100:

```
svm.linear.model2 <- svm(class ~., data = D_train, kernel = 'linear', cost = 100)
```

同样, 可以设置一个高斯核, 并将 `gamma` 参数设为 0.01:

```
svm.radial.model1 <- svm(class ~., data = D_train, kernel = 'radial', cost = 1,
gamma = 1e-2)
```

可通过如下代码改变 `gamma` 的值为 1:

```
svm.radial.model2 <- svm(class ~., data = D_train, kernel = 'radial', cost = 1,
gamma = 1)
```

最后可设置多项式核, 并将 `degree` 设为 2 或 3:

```
svm.poly.model1 <- svm(class ~., data = D_train, kernel = 'polynomial', degree
= 2, cost = 1)
```

```
svm.poly.model2 <- svm(class ~., data = D_train, kernel = 'polynomial', degree
= 3, cost = 1)
```

在我们提供的示例代码中, 将整个数据集随机分为训练集和测试集, 并通过设置不同的参数来学习得到不同的支持向量机分类器。这些分类器在测试集上的准确率也在运行时打印出来。

在分类中, 用得最多且一般来说效果最好的核是高斯核。而且使用高斯核需要调节的参数也较少: `cost` 值和 `gamma` 值。对于这两个参数值的设定, 我们建议首先通过交叉检验来确定 `cost` 的值。例如, 我们可以将 `cost` 的值设为 1~1000 之间 (当然可以更大, 因为参数的设置完全依赖于数据), 然后选出最优的 `cost` 值。确定好 `cost` 的值之后, 我们再试验几个不同的 `gamma` 值来找到最优的 `gamma` 值。当然, 想要得到最优的参数设置组合, 我们推荐格搜索 (grid search)。在 `e1071` 包中, 可以使用包中提供的 `tune.svm` 函数完成格搜索。在下面的例子中, 我们使用 `tune.svm` 寻找高斯核的最优参数:

```
tune.obj <- tune.svm(class~., data = D_train,
                    gamma = c(1e-4, 1e-3, 1e-2),
                    cost = c(1, 10))
print('summary of the cross-validation')
print(summary(tune.obj))
# Extract the optimal parameter value
gamma_best <- tune.obj$best.parameters$gamma
cost_best <- tune.obj$best.parameters$cost
```

但要注意, 求解高斯核对应的优化问题的计算复杂度较大, 因此, 在实际使用支持向量机时, 要根据数据的规模和可以允许的计算时间选择合理的参数。

6.5 损失函数和不同的分类算法

在前面的讨论中我们已经接触过不同的损失函数，如交叉熵损失函数和 Hinge 损失函数。损失函数在机器学习中非常重要，它是区别各个分类算法的核心因素。各个分类算法的差异中最突出的就是所优化的损失函数不同。

在机器学习的很多算法中，最小化的目标函数 F 通常为如下形式：

$$F = L + \lambda \text{Reg} \quad (6-75)$$

其中 L 是损失函数； Reg 是正则化项 (regularization)，用来控制所得模型的复杂度以避免所得的模型在训练集上过拟合； $\lambda \geq 0$ 用于控制损失函数和正则化项的权重。

在前面关于回归和分类算法的讨论中，很多算法都可以归入这一框架：

- 在 Lasso 中，损失函数是平方和损失函数，而正则化项是 L_1 范数；
- 在岭回归中，损失函数也是平方和函数，但正则化项是 L_2 范数；
- 在逻辑回归中，损失函数是交叉熵损失函数，正则化项是 L_2 范数（也可以加入 L_1 范数）；
- 在支持向量机中，损失函数是 Hinge 损失函数，正则化项是 L_2 范数。

在本节中，我们专门介绍分类算法中常用的各种损失函数。同时，也结合回归算法讨论一下不同的正则化项。在下面的讨论中，我们考虑数据集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ ，这里 $\mathbf{x}_i \in \mathbb{R}^d (i=1, 2, \dots, n)$ 是特征， $y_i \in \{-1, 1\}$ 是对应的类标。注意，这里为了讨论方便，假设 y_i 的取值是 -1 或者 1 。假设考虑的模型表示为 f ，对于 \mathbf{x}_i 的预测值记为 $f(\mathbf{x}_i)$ 。在损失函数中，因为我们通常直接考虑 y 与 $f(\mathbf{x})$ 的关系，所以通常也将 L 记为 $L(y, f)$ 。

6.5.1 损失函数

在分类问题中，定义在数据集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ 上的损失函数可以表示为：

$$L(y, f) = \sum_{i=1}^n L(y_i, f(\mathbf{x}_i)) \quad (6-76)$$

换言之，可以将损失函数分解为在每个样本点上的损失。下面我们逐一介绍对于样本点 (\mathbf{x}_i, y_i) 的常用损失函数，并将 L 表示为关于 $yf(\mathbf{x})$ 的函数，包括：

- 0-1 损失函数；
- Hinge 损失函数；
- 交叉熵损失函数；
- 指数损失函数；
- 平方和损失函数。

为什么在分类问题中要将损失函数 L 表示为关于 $yf(\mathbf{x})$ 的函数呢？我们知道，在回归问题中我们感兴趣的是残差 $y - f(\mathbf{x})$ ，因此经常将损失函数表示为关于残差的函数。在分类问

题中, 如果类标 y 表示为1或者-1的话, 那么 $yf(\mathbf{x})$ 就相当于回归问题中的 $y - f(\mathbf{x})$ 。进一步讲, 如果 $yf(\mathbf{x}) > 0$, 那么表示 \mathbf{x} 被 $f(\mathbf{x})$ 成功分类; 如果 $yf(\mathbf{x}) \leq 0$, 那么表示 \mathbf{x} 被 $f(\mathbf{x})$ 错误分类。在损失函数的设计中, 应该给错分的样本更大的惩罚值。从直观上讲, 在分类问题中, 损失函数应该对负值的 $yf(\mathbf{x})$ 给予更多的惩罚。

下面逐一介绍各个损失函数。

首先, 在分类问题中, 我们经常用来评价分类算法性能的准确率对应着 0-1 损失函数, 记为 $L_{01}(y, f(\mathbf{x}))$ 。严格地讲, 对于样本 (\mathbf{x}_i, y_i) 和 $f(\mathbf{x}_i)$, 0-1 损失函数 $L_{01}(y_i, f(\mathbf{x}_i))$ 定义为:

$$L_{01}(y_i, f(\mathbf{x}_i)) = \begin{cases} 0, & \text{如果 } y_i = \text{sign}(f(\mathbf{x}_i)) \\ 1, & \text{如果 } y_i \neq \text{sign}(f(\mathbf{x}_i)) \end{cases} \quad (6-77)$$

这里 sign 是符号函数, 其定义为:

$$\text{sign}(x) = \begin{cases} -1, & \text{如果 } x < 0 \\ 0, & \text{如果 } x = 0 \\ 1, & \text{如果 } x > 0 \end{cases} \quad (6-78)$$

注意, y_i 的取值是 -1 或者 1, 因此可以将 $L_{01}(y_i, f(\mathbf{x}_i))$ 记为关于 $y_i f(\mathbf{x}_i)$ 的函数:

$$L_{01}(y_i, f(\mathbf{x}_i)) = \begin{cases} 0, & \text{如果 } y_i f(\mathbf{x}_i) > 0 \\ 1, & \text{如果 } y_i f(\mathbf{x}_i) \leq 0 \end{cases} \quad (6-79)$$

在支持向量机中, 使用了 Hinge 函数, 其定义为:

$$L_{\text{hinge}}(y_i, f(\mathbf{x}_i)) = \max(0, 1 - y_i f(\mathbf{x}_i)) = (1 - y_i f(\mathbf{x}_i))_+ \quad (6-80)$$

这里记 $x_+ = \max(0, x)$ 。

在逻辑回归中, 使用的交叉熵损失函数也可以写为关于 $yf(\mathbf{x})$ 的函数。这里我们假设 $y_i \in \{-1, 1\}$, 但要注意我们在关于交叉熵的讨论中假设类标的取值是 0 或者 1。我们使用 y'_i 表示取值为 0 或者 1 的类标, 则有:

$$y'_i = (y_i + 1)/2 \quad (6-81)$$

根据原来的讨论, 使用类标 y'_i , 交叉熵损失函数为:

$$\begin{aligned} L_{\text{ce}} &= -y'_i \ln \frac{1}{1 + \exp(-f(\mathbf{x}_i))} - (1 - y'_i) \ln \left(1 - \frac{1}{1 + \exp(-f(\mathbf{x}_i))} \right) \\ &= y'_i \ln(1 + \exp(-f(\mathbf{x}_i))) - (1 - y'_i) \ln \left(\frac{1}{1 + \exp(f(\mathbf{x}_i))} \right) \\ &= y'_i \ln(1 + \exp(-f(\mathbf{x}_i))) + (1 - y'_i) \ln(1 + \exp(f(\mathbf{x}_i))) \end{aligned} \quad (6-82)$$

这里 $f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i$ 。因此, 当 $y_i = 1$ 时, $y'_i = 1$, $L_{\text{ce}} = \ln(1 + \exp(-f(\mathbf{x}_i)))$; 当 $y_i = -1$ 时, $y'_i = 0$, $L_{\text{ce}} = \ln(1 + \exp(f(\mathbf{x}_i)))$ 。综合考虑两种情况, 可将 L_{ce} 表示为:

$$L_{ce} = \ln(1 + \exp(-y_i f(\mathbf{x}_i))) \quad (6-83)$$

在线性回归中，我们使用的损失函数是平方和损失函数 $L_2(y, f(\mathbf{x}))$ 。利用 y_i 的取值是 -1 或者 1（即 $y_i^2 = 1$ ），也可以将其写为关于 $y_i f(\mathbf{x}_i)$ 的函数：

$$\begin{aligned} L_2(y_i, f(\mathbf{x}_i)) &= (y_i - f(\mathbf{x}_i))^2 \\ &= (y_i^2 - y_i f(\mathbf{x}_i))^2 \\ &= (1 - y_i f(\mathbf{x}_i))^2 \end{aligned} \quad (6-84)$$

在 AdaBoost 中，我们使用的是指数损失函数（exponential loss function），其定义为：

$$L_{exp}(y_i, f(\mathbf{x}_i)) = \exp(-y_i f(\mathbf{x}_i)) \quad (6-85)$$

我们将在稍后的章节中详细解释 AdaBoost 及为何在 AdaBoost 中引入指数损失函数。

在上面的讨论中，我们将损失函数都写成了关于 $yf(\mathbf{x})$ 的函数。在图 6-19 中给出了这些损失函数的图像。图 6-19 中横坐标对应于 $yf(\mathbf{x})$ ，纵坐标则是相应的损失函数值。表 6-6 中给出了每个损失函数的具体定义。

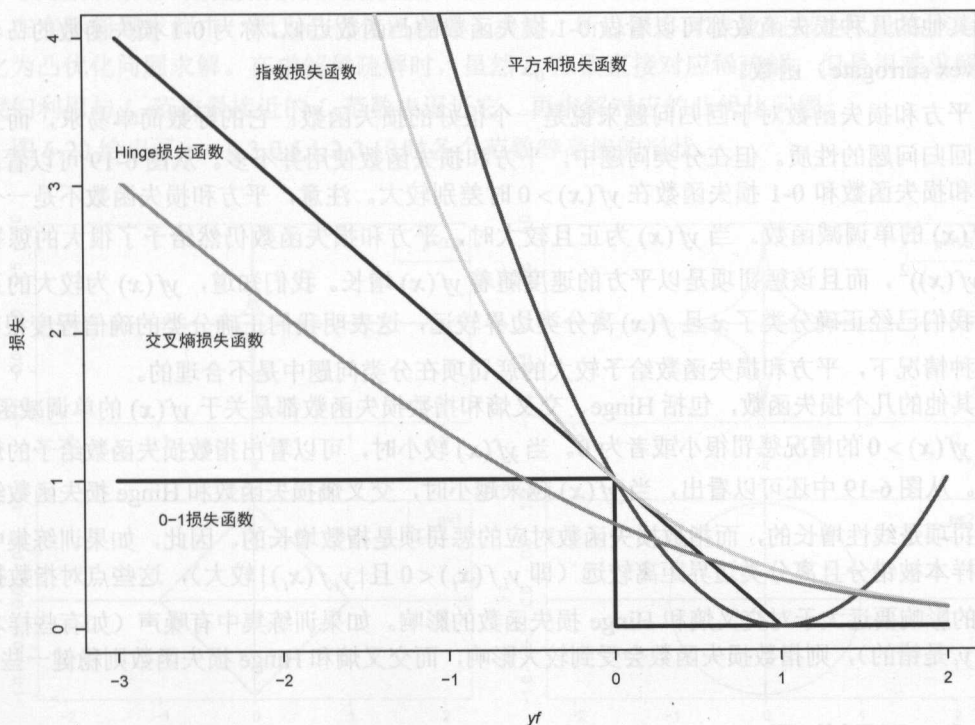


图 6-19 分类中常用的损失函数

表 6-6 分类中常用损失函数的具体定义

损失函数	定义
0-1 损失函数	$L_{01}(y, f(\mathbf{x})) = \begin{cases} 0, & \text{如果 } yf(\mathbf{x}) > 0 \\ 1, & \text{如果 } yf(\mathbf{x}) \leq 0 \end{cases}$
Hinge 损失函数	$L_{\text{hinge}}(y, f(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x}))$
交叉熵损失函数	$L_{\text{ce}}(y, f(\mathbf{x})) = \ln(1 + \exp(-yf(\mathbf{x})))$
指数损失函数	$L_{\text{exp}}(y, f(\mathbf{x})) = \exp(-yf(\mathbf{x}))$
平方和损失函数	$L_2(y, f(\mathbf{x})) = (1 - yf(\mathbf{x}))^2$

下面分析和比较一下各个损失函数的优缺点。

0-1 损失函数对应于准确率，是分类结果理想的衡量标准，但它最大的缺点是不连续且不可导。而在机器学习中，可导是一个非常重要的性质，原因在于在机器学习中，我们基本上都使用数值优化算法来求得最优解。如果目标函数本身不可导，就会导致所得的优化问题很难甚至不能求解。此外，当样本线性可分时，0-1 损失函数不能得到唯一解。从关于支持向量机的讨论可以发现，对于线性可分的训练集，我们可以构造无数个线性分类器对应于 0-1 损失函数为 0。因此，对于分类问题，我们基本上不直接使用 0-1 损失函数。

其他的几种损失函数都可以看做 0-1 损失函数的凸函数近似，称为 0-1 损失函数的凸替代 (convex surrogate) 函数。

平方和损失函数对于回归问题来说是一个很好的损失函数：它的导数简单易求，而且也符合回归问题的性质。但在分类问题中，平方和损失函数使用并不多。从图 6-19 可以看出，平方和损失函数和 0-1 损失函数在 $yf(\mathbf{x}) > 0$ 时差别较大。注意，平方和损失函数不是一个关于 $yf(\mathbf{x})$ 的单调减函数。当 $yf(\mathbf{x})$ 为正且较大时，平方和损失函数仍然给予了很大的惩罚项 $(1 - yf(\mathbf{x}))^2$ ，而且该惩罚项是以平方的速度随着 $yf(\mathbf{x})$ 增长。我们知道， $yf(\mathbf{x})$ 为较大的正值表示我们已经正确分类了 \mathbf{x} 且 $f(\mathbf{x})$ 离分类边界较远，这表明我们正确分类的确信程度很高。在这种情况下，平方和损失函数给予较大的惩罚项在分类问题中是不合理的。

其他的几个损失函数，包括 Hinge、交叉熵和指数损失函数都是关于 $yf(\mathbf{x})$ 的单调减函数，对于 $yf(\mathbf{x}) > 0$ 的情况惩罚很小或者为 0。当 $yf(\mathbf{x})$ 较小时，可以看出指数损失函数给予的惩罚较大。从图 6-19 中还可以看出，当 $yf(\mathbf{x})$ 越来越小时，交叉熵损失函数和 Hinge 损失函数给予的惩罚项是线性增长的，而指数损失函数对应的惩罚项是指数增长的。因此，如果训练集中有较多样本被错分且离分类边界距离较远（即 $y_i f(\mathbf{x}_i) < 0$ 且 $|y_i f(\mathbf{x}_i)|$ 较大），这些点对指数损失函数的影响要远大于对交叉熵和 Hinge 损失函数的影响。如果训练集中有噪声（如有些样本的类标 y_i 是错的），则指数损失函数会受到较大影响，而交叉熵和 Hinge 损失函数则稳健一些。

6.5.2 正则化项

在机器学习的很多算法中，我们使用正则化项来控制模型的复杂度。在回归分析中，我

我们已经分析了 L_1 范数和 L_2 范数。例如, Lasso 和岭回归都使用平方和损失函数, 但 Lasso 中的正则化项是 L_1 范数, 而岭回归中的正则化项是 L_2 范数。在支持向量机中, 我们知道所对应的损失函数是 Hinge 函数, 而正则化项实际对应于分类间隔。通过最大化分类间隔, 我们可以控制模型的复杂度。事实上, 从支持向量机中关于优化问题的推导我们可以看出, 最大化分类间隔等价于最小化 $\|w\|_2^2$ 。换言之, 支持向量机中的正则化项还是 L_2 范数。并且在支持向量机的例子中, 我们清楚地看到最小化 L_2 范数和控制模型复杂度的直接联系。

事实上, 对于向量 $w = [w_1, w_2, \dots, w_d]^T$, 在第 3 章我们已经定义了广义的 L_p 范数:

$$L_p(w) = \left(\sum_{i=1}^d |w_i|^p \right)^{\frac{1}{p}} \quad (6-86)$$

当 $p \geq 1$ 时, $L_p(w)$ 是凸函数, 因此是良定义的范数。当 $0 \leq p < 1$ 时, 上面 $L_p(w)$ 的定义不是良定义的范数, 但很多时候大家还是直接称之为 L_p 范数。特别指出, 当 $p = 0$ 时, $L_0(w)$ 定义为:

$$L_0(w) = |\{i: w_i \neq 0\}| \quad (6-87)$$

即 $L_0(w)$ 定义 w 中非零元素的数目。如果使用 $L_0(w)$ 作为正则化项, 我们能够直接最小化 w 中非零元素的数目, 所得的解称为稀疏解, 能够在做回归或者分类的时候进行特征选取。

在实际问题求解中, 因为凸优化问题易于求解, 所以很多时候我们都是尽量将原始问题转化为凸优化问题求解。在求解稀疏解时, 虽然 L_0 范数直接对应稀疏解, 但是很难求解, 一般我们利用与 L_0 范数最接近的 L_1 范数去逼近它, 再求解对应的凸优化问题。

图 6-20 给出了 $p = 0.3, 0.5, 1, 2, 10$ 时各个范数等高线的形状。

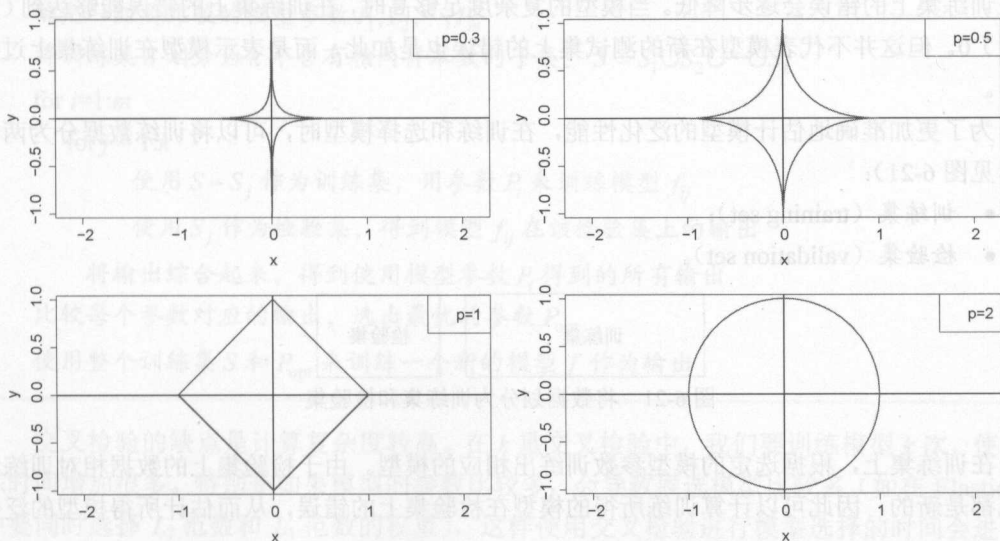


图 6-20 不同的正则化项对应范数的等高线

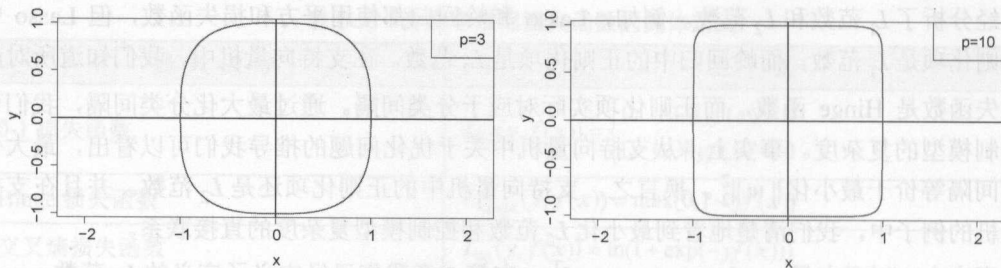


图 6-20 不同的正则化项对应范数的等高线 (续)

6.6 交叉检验和 caret 包

本节首先介绍模型选择 (model selection) 和交叉检验 (cross-validation), 然后重点介绍 R 中的 caret 包^①。caret 包实现了多种机器学习算法的参数估计, 包括使用交叉检验来进行模型选择, 是实际应用中非常受欢迎的软件包。

6.6.1 模型选择和交叉检验

在使用上面讨论的机器学习算法构建模型时, 对于同样的算法, 使用不同的参数会得到不同的模型。在模型选择中, 我们要估计不同模型的性能并从中选出最优的模型。这里的最优指的是模型对新数据的泛化性能最优。在给定训练集的情况下, 模型在训练集上的错误并不是该模型泛化性能的良好估计。从回归算法的讨论可知, 当我们增加模型的复杂度时, 模型在训练集上的错误会逐步降低。当模型的复杂度足够高时, 在训练集上的错误能够达到 (或接近) 0。但这并不代表模型在新的测试集上的错误也是如此, 而是表示模型在训练集上过拟合了。

为了更加准确地估计模型的泛化性能, 在训练和选择模型时, 可以将训练数据分为两部分 (见图 6-21):

- 训练集 (training set);
- 检验集 (validation set)。

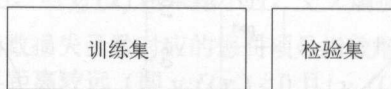


图 6-21 将数据划分为训练集和检验集

在训练集上, 根据选定的模型参数训练出相应的模型。由于检验集上的数据相对训练集来说都是新的, 因此可以计算训练所得的模型在检验集上的错误, 从而估计所得模型的泛化

^① <https://cran.r-project.org/web/packages/caret/index.html>

性能。在实际中，可以使用 2/3 的数据来训练样本，而使用剩余的 1/3 作为检验集来估计模型的性能。

在数据不是很充足时，为了最大程度地利用已有的数据，交叉检验是一个更好的选择。在数据较为有限的情况下，检验集通常都较小，使用检验集来估计模型的性能可能会存在偏差。而利用交叉检验，整个训练集都可以作为检验集，同时也能使用整个训练集来训练模型，因此对于模型性能的估计会更加准确。

具体来说，在 k 重交叉检验 (k -fold cross-validation) 中，将训练集 S 划分为互不重叠的 k 个子集 S_1, S_2, \dots, S_k ，如图 6-22 所示。一般而言，每个 S_i 中包括相同数目的样本。

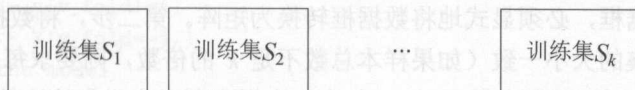


图 6-22 k 重交叉检验中数据的划分

在 k 重交叉检验中，顺次从 S_1, S_2, \dots, S_k 选出一个作为检验集，而将剩余的数据作为训练集训练模型，并计算该模型在检验集上的预测结果。该过程重复 k 次，使得每个 S_i 都被选为检验集一次。这样，我们就将整个训练集用作了检验集，从而得到模型的性能估计。根据交叉检验选定最优的模型参数后，我们使用整个训练集重新训练一个模型作为最终的输出。在实际中，根据数据的大小，可以将 k 选为 5、10 等值。算法 6-2 给出了使用交叉检验进行模型选择的具体过程。

算法 6-2 使用交叉检验进行模型选择

确定一组要比较的模型参数 P_1, P_2, \dots, P_m

将训练集 S 划分为 k 个含有相同样本数的子集： $S = S_1 \cup S_2 \cup \dots \cup S_k$

for $i=1:m$

for $j=1:k$

使用 $S - S_j$ 作为训练集，用参数 P_i 来训练模型 f_{ij}

使用 S_j 作为检验集，得到模型 f_{ij} 在该检验集上的输出

将输出综合起来，得到使用模型参数 P_i 得到的所有输出

比较每个参数对应的输出，选出最优的参数 P_{opt}

使用整个训练集 S 和 P_{opt} 来训练一个新的模型 f 作为输出

交叉检验的缺点是计算复杂度较高。在 k 重交叉检验中，我们要训练模型 k 次，使得训练时间增加很多。特别是如果模型的参数比较多，会导致候选模型比较多（如在 Elastic Net 中要同时选择 L_2 范数和 L_1 范数的权重），这样使用交叉检验进行模型选择的时间会进一步提高。

6.6.2 在 R 中实现交叉检验以及 caret 包

本节介绍如何在 R 中利用交叉检验进行模型选择，并重点介绍 caret 软件包。

在介绍 caret 包之前，以 glmnet 为例，说明如何在 R 中进行交叉检验。虽然 glmnet 包中的 cv.glmnet 函数可以实现交叉检验，但是这里我们的重点在于说明交叉检验的一般流程。此外，在这个例子中，我们只考虑 L_2 范数的权重 lambda。完全的 R 程序见文件 CV_example.R。下面我们讲解其中的关键部分。

第一步，我们准备好数据 X 和 y ，这里 X 是矩阵， y 是一个向量。注意，使用 glmnet 时输入数据不能是数据框，必须显式地将数据框转换为矩阵。第二步，将数据随机分成 k 个子集，且保证每个子集的大小一致（如果样本总数不是 k 的倍数，则要求每个子集的大小尽量接近）。在下面的代码中，使用向量 v_fold 包含每个样本所对应子集的编号，使用 sample(n) 产生一个 $1 \sim n$ 的随机排列。这里使用 set.seed 函数设定随机数生成器的种子，以便我们重复实验结果。第三步，对于每个 lambda 参数，使用交叉检验来训练模型，并计算在检验集上的预测值，最后将结果保存在矩阵 Y_valid 中。第四步，计算每个 lambda 值对应的准确率，选择最优的 lambda 值并利用整个训练集重新训练一个模型。在这个例子中，我们考虑了 9 个 lambda 的值，包括 0、0.001、0.005、0.01、0.02、0.03、0.05、0.1、1。在我们的实验结果中，lambda_optimal=0.01，最后的模型保存在 M_optimal 中。

```
library(glmnet)

# Step 1. Load the data and set parameters
f_in <- 'Data/pima-indians-diabetes.csv'
D <- read.csv(f_in)
D$class <- as.factor(D$class)
y <- D$class
X <- D
X$class <- NULL
X <- as.matrix(X)
k <- 5

# Step 2. Split the data into k folds
n <- nrow(D)
set.seed(0)
v_random <- sample(n)
v_fold <- rep(1, n)
sample_size_per_fold <- floor(n/k)
for (i in 1:k) {
  range_min = (i-1) * sample_size_per_fold + 1
  range_max = i * sample_size_per_fold
  v_fold[(v_random>=range_min) & (v_random<=range_max)] <- i
}
```

```

# Step 3. Perform cross-validation for several logistic regression models
lambda_list <- c(0, 0.001, 0.005, 0.01, 0.02, 0.03, 0.05, 0.1, 1)
param_num <- length(lambda_list)

# We use a matrix to score the prediction on the validation data set
Y_valid <- matrix(0, n, param_num)
for (i in 1:param_num) {
  lambda <- lambda_list[i]
  y_valid <- rep(0, n)
  for (j in 1:k) {
    X_train <- X[v_fold!=j, ]
    y_train <- y[v_fold!=j]
    X_valid <- X[v_fold==j, ]
    # Train the model
    Mj <- glmnet(X_train, y_train, family = 'binomial', lambda = lambda)
    # Make prediction on the validation data set
    y_valid_j <- predict(Mj, X_valid, type='class')
    y_valid[v_fold==j] <- y_valid_j
  }
  Y_valid[, i] <- y_valid
}

# Step 4. Pick the optimal lambda and rebuild the model using the whole training
# data set
accuracy_list <- rep(0, param_num)
for (i in 1:param_num) {
  accuracy_list[i] <- sum(Y_valid[,i]==y) / n
}
lambda_optimal <- lambda_list[which.max(accuracy_list)]
msg = paste0('lambda_optimal = ', lambda_optimal)
print(msg)
M_optimal <- glmnet(X, y, family = 'binomial', lambda = lambda_optimal)

```

下面具体介绍 caret 包，并使用具体的例子讨论如何利用 caret 包训练逻辑回归和决策树。

R 是开源的，虽然 R 中的资源和各种软件非常丰富，但是也比较杂乱。例如，我们前面介绍的 rpart 包和 glmnet 的接口就不一样。在 rpart 包中，我们可以直接使用保存为数据框的输入数据，但在 glmnet 中，输入数据只能是矩阵而不能是数据框。R 中的 caret (classification and regression training) 包为 R 中超过 200 个包提供了统一的接口，包括前面介绍的 glmnet、rpart、e1071 等，也包括第 9 章将要介绍的 randomForest 和 gbm 等包。caret 目前支持的包的完全列表可以在 caret 的主页查到^①。

使用 caret 包，我们可以很简单地建立、评价和比较多个模型。caret 提供了包括交叉检验在内的多种模型选择的方法。使用 caret 包，交叉检验的过程将会变得很简单。此外，caret 还提供了数据预处理的一些功能。

① <http://topepo.github.io/caret/modelList.html>

具体来说,在 `caret` 包中,常用的函数包括 `train` 和 `predict`,此外还有一个重要的辅助函数 `trainControl`。

利用 `train` 函数,可以直接指定训练数据、挑选模型的方法(如交叉检验)、挑选模型的指标(如回归中的 `RMSE` 和 R^2 ,分类问题中的准确率和 `ROC` 等)、调用的包(如 `glmnet` 或者 `rpart`)、交叉检验中要测试的参数等。这些参数有一些可以在 `train` 函数中直接设置,有些需要在 `trainControl` 中设置。`train` 函数的逻辑流程与算法 6-2 一致。得到最终选定的参数后,`caret` 使用所有的数据重新训练一个模型并返回。之后我们可以直接使用 `predict` 函数(实质是 `caret` 包中的 `predict.train`)函数处理新数据。

`train` 函数的主要参数如下。

- `method`: 一个字符串,用于指定要调用的模型。使用 `names(getModelInfo())` 可以得到当前 `caret` 包中所支持的所有模型的列表。
- `metric`: 也是字符串,用来指定在模型选择中要优化的指标。在分类中可以是 'Accuracy' 或 'ROC',在回归中可以是 'RMSE' 或 'Rsquared'。
- `trControl`: 一个列表,表示进行模型选择时的控制参数。通常,我们先使用 `trainControl` 得到一个列表,再赋给 `trControl`。
- `tuneGrid`: 一个数据框,表示在模型选择中我们要考虑的参数可能的取值。该数据框的列名必须与对应模型中的控制参数同名。如果不提供,则 `caret` 自动选择要遍历的参数值。
- `preProcess`: 一个字符串,表示对于数据的预处理操作,如 'center'、'scale' 等。在默认状态下,不进行任何预处理。

在指定输入数据时,同 R 中的其他很多包类似,`train` 支持两种模式: `train(x, y)` 和 `train(formula, data)`。

在 `train(x, y)` 中, `x` 表示输入数据(每行对应一个样本,每列对应一个特征), `y` 表示要预测的量。在 `train(formula, data)` 中, `data` 表示所有的数据(包括所有特征和要预测的量),而在 `formula` 中我们指定哪一列是要预测的量,以及要用哪些列来预测。

在使用 `trainControl` 函数时,通常要使用的主要参数有以下几个。

- `method`: 一个字符串,表示采用何种取样方法进行模型选择。最常用的是 'repeatedcv',表示进行多次交叉检验。
- `number`: 一个整数,表示进行交叉检验时的重数。
- `repeats`: 一个整数,表示进行 `repeatedcv` 时的重复次数。
- `classProbs`: `TRUE` 或者 `FALSE`,表示在分类模型中是否计算样本属于每个类别的概率。例如,在计算 `ROC` 时,我们就需要明确计算概率。

`predict` 函数的使用则较简单。假设 `M` 是前面 `train` 函数的返回结果, `D_test` 是新的测试数据,则我们可以使用

```
y_test <- predict(M, newdata=D_test)
```

来得到预测结果。

在 caret 中, 主要使用重取样 (resampling) 的方法来优化模型参数从而进行模型选择。重取样的思想是: 从整个训练集中采用某种取样方法得到一个新的数据集, 在该数据集上训练模型, 并将原始训练集剩余的样本作为检验集来计算模型的性能; 重复该过程多次, 从而找出最优的模型参数。在交叉检验中, 我们把整个训练集分成若干等份, 然后在每次重取样时去掉一个子集作为检验集, 剩余的数据作为训练集。在 caret 中, 还支持除交叉检验以外的其他重取样方法, 如每次可以进行 bootstrap 取样 (将在第 9 章讨论)。注意, 在重取样中, 一旦取样数据确定, 可以并行地训练多个模型以加快计算速度。在 R 中, 可以使用 foreach 包及相关的包来实现。例如, 对于多核计算机, 可以用 foreach 和 doMC 来利用多个核, 这两个包的具体使用读者可参阅相关文档^①。

下面使用 mlbench 中的 Sonar 数据集来介绍使用 caret 包进行交叉检验的具体例子。完全的 R 代码在文件 caret_example.R 中。在这个例子中, 首先安装相应的软件包, 包括 glmnet、rpart、caret 和 mlbench。接下来第一步, 使用 data(Sonar) 载入数据, 并使用 str(Sonar) 查看数据每列的类型。注意, 最后一列 Class 是因子类型, 表示样本的类别。

第二步, 使用 caret 包和 glmnet 包, 按照由简单到复杂的顺序依次建立 5 个逻辑回归模型。注意, Sonar 是一个数据框。在前面直接使用 glmnet 时, 需要将输入数据显式地转换为矩阵, 而通过 caret 调用 glmnet 时, 不需要显式转换。

在构建第一个逻辑回归模型 M_glmnet1 时, 首先使用 trainControl 函数构建一个列表对象: 指定 method 参数为 'repeatedcv', 表示使用多次交叉检验; 将 number 设为 5, 表示进行 5 重交叉检验; 将 repeats 设为 3, 表示交叉检验要运行 3 次。在使用 train 函数时, 指定输入数据为 Sonar, 且用公式 Class~. 表示利用所有其他列来预测 Class 列, 用 method='glmnet' 表示 caret 将调用 glmnet 包构建一个逻辑回归模型。

```
k = 5
cv_repeat_num <- 3
fitControl1 <- trainControl(method = 'repeatedcv',
                             number = k,
                             repeats = cv_repeat_num)
M_glmnet1 <- train(Class~.,
                   data = Sonar,
                   method = 'glmnet',
                   trControl = fitControl1)
```

在得到 train 函数返回的 M_glmnet1 对象后, 可以检查使用多次交叉检验之后的模型选择结果。

- M_glmnet1\$bestTune 返回选择的最优模型对应的参数。在 glmnet 中为对应的 alpha 值和 lambda 值。
- M_glmnet1\$method 返回模型对应的包, 这里是 glmnet。

^① <https://cran.r-project.org/web/packages/doMC/vignettes/gettingstartedMC.pdf>

- `M_glmnet1$metric` 返回在交叉检验中优化的性能指标，这里是分类中的准确率。
- `M_glmnet1$modelType` 返回模型的类型，这里是分类。
- `M_glmnet1$results` 返回在交叉检验中不同的参数对应的性能指标（这里包括准确率等）。

上面介绍了 `M_glmnet1` 的主要成员，读者可以使用 `attributes(M_glmnet1)` 查看 `M_glmnet1` 中的所有成员。

在这个例子中，对应的输出如下：

```
> # Check the optimal model parameter of the selected model via cross-validation
> M_glmnet1$bestTune
      alpha      lambda
3    0.1 0.04318733
> # Check the model name, here it is glmnet
> M_glmnet1$method
[1] "glmnet"
> # Check the metric to be optimized, here it is Accuracy
> M_glmnet1$metric
[1] "Accuracy"
> # Check the model type, here it is classification
> M_glmnet1$modelType
[1] "Classification"
> # Check the performance (accuracy) for different parameter values
> M_glmnet1$results
      alpha      lambda Accuracy      Kappa AccuracySD      KappaSD
1 0.10 0.0004318733 0.7581107 0.5115405 0.07082978 0.1444585
2 0.10 0.0043187332 0.7595819 0.5157644 0.07147509 0.1438111
3 0.10 0.0431873324 0.7627952 0.5227198 0.06249494 0.1252137
4 0.55 0.0004318733 0.7596980 0.5142755 0.07231546 0.1470977
5 0.55 0.0043187332 0.7562911 0.5085341 0.07837867 0.1580675
6 0.55 0.0431873324 0.7434766 0.4834793 0.08075096 0.1611272
7 1.00 0.0004318733 0.7613628 0.5169023 0.07624527 0.1553086
8 1.00 0.0043187332 0.7548587 0.5062906 0.07481600 0.1506429
9 1.00 0.0431873324 0.7467286 0.4909440 0.06744425 0.1336721
```

在第二个逻辑回归模型中，指定了在交叉检验中要比较的不同参数值。我们在 `lambda_list` 和 `alpha_list` 中指定所有可能的 `lambda` 值和 `alpha` 值；然后使用 `expand.grid` 函数生成一个数据框 `myParamGrid`，其中包含 `lambda` 和 `alpha` 的所有组合。注意，该数据框的列名必须要与对应模型中的控制参数同名。在使用 `train` 函数时，指定 `tuneGrid` 参数为 `myParamGrid` 即可。

```
lambda_list <- c(0, 0.001, 0.005, 0.01, 0.02, 0.03, 0.05, 0.1, 1)
alpha_list <- c(0, 0.1)
myParamGrid <- expand.grid(lambda=lambda_list, alpha=alpha_list)
fitControl2 <- trainControl(method = 'repeatedcv',
                             number = k,
                             repeats = cv_repeat_num)
M_glmnet2 <- train(Class~.,
```

```
data = Sonar,
method = 'glmnet',
trControl = fitControl2,
tuneGrid = myParamGrid)
```

在这个例子中，使用 `expand.grid` 生成的 `myParamGrid` 数据框如下：

```
> myParamGrid
  lambda alpha
1  0.000   0.0
2  0.001   0.0
3  0.005   0.0
4  0.010   0.0
5  0.020   0.0
6  0.030   0.0
7  0.050   0.0
8  0.100   0.0
9  1.000   0.0
10 0.000   0.1
11 0.001   0.1
12 0.005   0.1
13 0.010   0.1
14 0.020   0.1
15 0.030   0.1
16 0.050   0.1
17 0.100   0.1
18 1.000   0.1
```

与第一个模型类似，我们检查所得最优模型，其对应的输出如下。可以看出，在给定 `alpha` 和 `lambda` 值的情况下，根据 `M_glmnet2$bestTune`，最优组合是 `alpha=0`、`lambda=0.1`。同时，`M_glmnet2$results` 给出了不同的 `alpha` 和 `lambda` 组合对应的模型性能。

```
> # Check the optimal model parameter of the selected model via cross-validation
> M_glmnet2$bestTune
  alpha lambda
8     0     0.1
> # Check the performance (accuracy) for different parameter values
> M_glmnet2$results
  alpha lambda Accuracy Kappa AccuracySD KappaSD
1  0.0  0.000 0.7773123 0.5517966 0.06008416 0.1215905
2  0.0  0.001 0.7773123 0.5517966 0.06008416 0.1215905
3  0.0  0.005 0.7773123 0.5517966 0.06008416 0.1215905
4  0.0  0.010 0.7773123 0.5517966 0.06008416 0.1215905
5  0.0  0.020 0.7773123 0.5517966 0.06008416 0.1215905
6  0.0  0.030 0.7821516 0.5614362 0.05702015 0.1160416
7  0.0  0.050 0.7804869 0.5587645 0.05216941 0.1051730
8  0.0  0.100 0.7821885 0.5619241 0.05439456 0.1104805
9  0.0  1.000 0.7615851 0.5185915 0.06908800 0.1407203
10 0.1  0.000 0.7532318 0.5028267 0.06820656 0.1382080
```



```

11 0.1 0.001 0.7660112 0.5292147 0.06393240 0.1282588
12 0.1 0.005 0.7691489 0.5357482 0.06823736 0.1379260
13 0.1 0.010 0.7789788 0.5547546 0.05820813 0.1179553
14 0.1 0.020 0.7821147 0.5619627 0.05340633 0.1072235
15 0.1 0.030 0.7756494 0.5488414 0.05623293 0.1132750
16 0.1 0.050 0.7724343 0.5426609 0.05394738 0.1082715
17 0.1 0.100 0.7630248 0.5243107 0.05814217 0.1154458
18 0.1 1.000 0.7355743 0.4583782 0.04822279 0.1012266

```

在前两个逻辑回归模型的交叉检验中，我们都最大化了准确率。在第三个逻辑回归模型中，我们要最大化 AUC (Area Under ROC Curve)。AUC 也是分类中常用的一个性能指标，6.7 节会详细介绍。在计算 AUC 时，我们需要知道样本属于每个类的概率。因此，在 `trainControl` 函数中，可以将 `classProbs` 参数设为真，同时将计算预测结果好坏的函数置为 `caret` 包中的 `twoClassSummary` 函数。在调用 `train` 函数时，还要显式地将 `metric` 指定为 'ROC'。之后检查所得的最优参数、交叉检验中调用的包和要优化的指标，以及各个参数值对应的算法性能。对应的 R 代码如下：

```

fitControl3 <- trainControl(method = 'repeatedcv',
                             number = k,
                             repeats = cv_repeat_num,
                             classProbs = T,
                             summaryFunction = twoClassSummary)
M_glmnet3 <- train(Class~.,
                   data = Sonar,
                   method = 'glmnet',
                   trControl = fitControl3,
                   metric='ROC')
# Check the optimal model parameter of the selected model via cross-validation
M_glmnet3$bestTune
# Check the model name
M_glmnet3$method
# Check the metric to be optimized
M_glmnet3$metric
# Check the performance (accuracy) for different parameter values
M_glmnet3$results

```

对应的输出如下。可以看出，`M_glmnet3$metric` 是 'ROC'。此外，`M_glmnet3$results` 的输出也与前面两个例子不同：没有计算 Accuracy，但输出了 ROC。我们将在 6.7 节详细介绍 sensitivity (Sens)、specificity (Spec) 等性能指标。

```

> # Check the optimal model parameter of the selected model via cross-validation
> M_glmnet3$bestTune
  alpha      lambda
3  0.1 0.04318733
> # Check the model name
> M_glmnet3$method
[1] "glmnet"

```

```
> # Check the metric to be optimized
> M_glmnet3$metric
[1] "ROC"
> # Check the performance (accuracy) for different parameter values
> M_glmnet3$results
```

	alpha	lambda	ROC	Sens	Spec	ROCSD	SensSD	SpecSD
1	0.10	0.0004318733	0.8250607	0.7595520	0.7194737	0.05394260	0.09463985	0.10113034
2	0.10	0.0043187332	0.8454164	0.7805007	0.7298246	0.04466097	0.09994404	0.09204975
3	0.10	0.0431873324	0.8689009	0.8069829	0.7405263	0.04560518	0.10710990	0.10737911
4	0.55	0.0004318733	0.8234772	0.7624506	0.7194737	0.05639064	0.11546886	0.10867497
5	0.55	0.0043187332	0.8483690	0.7864295	0.7403509	0.04386016	0.09338402	0.09333055
6	0.55	0.0431873324	0.8472582	0.7801054	0.7300000	0.04503293	0.08608097	0.10888869
7	1.00	0.0004318733	0.8199213	0.7594203	0.7091228	0.05695943	0.11890709	0.11820309
8	1.00	0.0043187332	0.8488787	0.7952569	0.7266667	0.04297800	0.09286074	0.09309281
9	1.00	0.0431873324	0.8404629	0.7864295	0.7292982	0.04698343	0.08865779	0.08191700

在第四个逻辑回归模型中，在交叉检验中最大化 AUC，但同时使用我们自己确定的 alpha 和 lambda 值。与第三个模型相比，在 train 函数中将 tuneGrid 参数指定为 myParamGrid 即可。

```
fitControl4 <- trainControl(method = 'repeatedcv',
                             number = k,
                             repeats = cv_repeat_num,
                             classProbs = T,
                             summaryFunction = twoClassSummary)

M_glmnet4 <- train(Class~.,
                   data = Sonar,
                   method = 'glmnet',
                   trControl = fitControl4,
                   tuneGrid = myParamGrid,
                   metric='ROC')

# Check the optimal model parameter of the selected model via cross-validation
M_glmnet4$bestTune
# Check the model name
M_glmnet4$method
# Check the metric to be optimized
M_glmnet4$metric
# Check the performance (accuracy) for different parameter values
M_glmnet4$results
```

对应的输出如下。从 M_glmnet4\$results 可以看出，最优的 alpha 值和 lambda 值对应最高的 ROC 值。

```
> # Check the optimal model parameter of the selected model via cross-validation
> M_glmnet4$bestTune
  alpha lambda
8      0    0.1
> # Check the model name
> M_glmnet4$method
[1] "glmnet"
```

```

> # Check the metric to be optimized
> M_glmnet4$metric
[1] "ROC"
> # Check the performance (accuracy) for different parameter values
> M_glmnet4$results

```

	alpha	lambda	ROC	Sens	Spec	ROCSD	SensSD	SpecSD
1	0.0	0.000	0.8771347	0.8052701	0.7564912	0.05540251	0.09926736	0.10691776
2	0.0	0.001	0.8771347	0.8052701	0.7564912	0.05540251	0.09926736	0.10691776
3	0.0	0.005	0.8771347	0.8052701	0.7564912	0.05540251	0.09926736	0.10691776
4	0.0	0.010	0.8771347	0.8052701	0.7564912	0.05540251	0.09926736	0.10691776
5	0.0	0.020	0.8771347	0.8052701	0.7564912	0.05540251	0.09926736	0.10691776
6	0.0	0.030	0.8779592	0.8051383	0.7600000	0.05733968	0.09330731	0.10139607
7	0.0	0.050	0.8774225	0.7960474	0.7635088	0.05785979	0.09384216	0.10695939
8	0.0	0.100	0.8781121	0.8077734	0.7738596	0.05964284	0.08636892	0.12334968
9	0.0	1.000	0.8590198	0.8227931	0.7187719	0.05797275	0.05535949	0.11722819
10	0.1	0.000	0.8399761	0.7898551	0.6910526	0.04417983	0.08314388	0.08165652
11	0.1	0.001	0.8514229	0.7989460	0.6945614	0.04598263	0.08842463	0.07543874
12	0.1	0.005	0.8655700	0.8077734	0.7356140	0.05232582	0.09452718	0.09316717
13	0.1	0.010	0.8707312	0.7992095	0.7461404	0.05523448	0.09542990	0.09928946
14	0.1	0.020	0.8753308	0.8023715	0.7598246	0.05758868	0.09610051	0.10410294
15	0.1	0.030	0.8768705	0.7963109	0.7633333	0.05776288	0.09636278	0.09829606
16	0.1	0.050	0.8744848	0.8021080	0.7668421	0.06007231	0.09227325	0.10160368
17	0.1	0.100	0.8680854	0.7869565	0.7701754	0.05787892	0.07471969	0.10861973
18	0.1	1.000	0.8212329	0.8977602	0.5819298	0.05607401	0.04997118	0.10050263

在第五个逻辑回归模型中，首先，使用 `caret` 包中的 `createDataPartition` 函数产生训练集对应的样本在数据 `Sonar` 中的行号（保存在 `trainingIndex` 中）。`createDataPartition` 函数主要用于分层抽样（stratified sampling）：在分类问题中按照正类和负类样本的比例取样，使得抽样取得的样本集中正负类样本的比例保持不变。`createDataPartition` 函数中的参数 `y` 表示总体中的类标列，`p` 表示取样比例，`list` 表示返回的结果是否是列表。然后，利用 `trainingIndex` 得到训练集 `D_train` 和测试集 `D_test`。在利用交叉检验得到模型 `M_glmnet5` 之后，调用 `predict` 函数得到测试集上的预测类别。

```

trainingIndex <- createDataPartition(y=Sonar$Class, p=0.6, list=F)
D_train <- Sonar[trainingIndex,]
D_test <- Sonar[-trainingIndex,]
fitControl5 <- trainControl(method = 'repeatedcv',
                             number = k,
                             repeats = cv_repeat_num)
M_glmnet5 <- train(Class~.,
                   data = D_train,
                   method = 'glmnet',
                   trControl=fitControl5)
y_test <- predict(M_glmnet5, newdata = D_test)

```

第三步，使用 `caret` 训练一个 `rpart` 模型。在 `rpart` 中，`caret` 只支持 `cp`。在下面的 R 代码中，我们得到了两个决策树模型 `M_rpart1` 和 `M_rpart2`。在第一个模型中，主要使用 `train`

中的默认参数，在第二个模型中使用自定义的参数 `cp` 的值。与前面的 `glmnet` 模型相比，最主要的区别在于 `train` 函数中的 `method` 参数要设置为 `'rpart'`，同时生成数据框 `myParamGrid_rpart` 时要保证对应的列名为 `cp`。

具体代码如下：

```
# Train a rpart model using default parameters provided by caret and cross-validation
fitControl_rpart1 <- trainControl(method = 'repeatedcv',
                                   number = k,
                                   repeats = cv_repeat_num)

M_rpart1 <- train(Class~.,
                  data = Sonar,
                  method = 'rpart',
                  trControl = fitControl_rpart1)

# Check the optimal model parameter of the selected model via cross-validation
M_rpart1$bestTune
# Check the model name
M_rpart1$method
# Check the metric to be optimized
M_rpart1$metric
# Check the performance (accuracy) for different parameter values
M_rpart1$results

# Train a rpart model using provided parameter values and cross-validation
fitControl_rpart2 <- trainControl(method = 'repeatedcv',
                                   number = k,
                                   repeats = cv_repeat_num)

cp_list <- c(0, 0.01, 0.1, 1)
myParamGrid_rpart <- expand.grid(cp=cp_list)
M_rpart2 <- train(Class~.,
                  data = Sonar,
                  method = 'rpart',
                  trControl = fitControl_rpart2,
                  tuneGrid = myParamGrid_rpart)

# Check the optimal model parameter of the selected model via cross-validation
M_rpart2$bestTune
# Check the model name
M_rpart2$method
# Check the metric to be optimized
M_rpart2$metric
# Check the performance (accuracy) for different parameter values
M_rpart2$results
```

下面给出 `M_rpart2` 对应的输出：

```
> # Check the optimal model parameter of the selected model via cross-validation
> M_rpart2$bestTune
  cp
3 0.1
> # Check the model name
> M_rpart2$method
```



```
[1] "rpart"
> # Check the metric to be optimized
> M_rpart2$metric
[1] "Accuracy"
> # Check the performance (accuracy) for different parameter values
> M_rpart2$results
      cp Accuracy      Kappa AccuracySD      KappaSD
1 0.00 0.6910704 0.3757473 0.077546696 0.1567621
2 0.01 0.6910704 0.3757473 0.077546696 0.1567621
3 0.10 0.7114766 0.4143188 0.069656667 0.1376974
4 1.00 0.5336845 0.0000000 0.008177767 0.0000000
```

利用 `caret` 包，可以采用统一的接口简单地调用 R 中的 200 多个包，并使用不同的重取样方法来确定最优模型的参数。但在选择模型参数时，`caret` 已经事先选定了可以调节的模型参数。例如，在 `glmnet` 中就是参数 `alpha` 和 `lambda`，在 `rpart` 中只有一个参数 `cp`。如果要调节更多的参数，有两种选择：

- 继续使用 `caret` 包，但需要做更多的定制化工作；
- 不使用 `caret` 包，自己直接使用对应的包，并实现交叉检验的对应代码。

在一般情况下，如果模型所对应的包不是特别生僻，那么通常推荐第二种方法。

对于第二种方法，我们提供了在 `caret` 包中对 `gbm` 进行定制的示例。关于 `gbm` 的详细用法请参照 9.5.3 节。这里我们使用 `gbm` 作为定制 `caret` 包的一个例子。我们提供了 3 个程序：`caret_custom_example.R`、`gbm_CV_func.R` 和 `classification_measures.R`。其中 `classification_measures.R` 实现了多种不同的分类算法评价指标，包括多类分类算法的评价指标。`gbm_CV_func.R` 包详细讲解了如何在 `caret` 包中对 `gbm` 进行定制，包括如何在 `caret` 包中优化自己实现的算法评价指标。`caret_custom_example.R` 文件讲解了如何调用 `gbm_CV_func.R` 中的函数在实际数据上进行交叉检验。感兴趣的读者可以阅读并运行相关程序。

6.7 分类算法的评价和比较

在上一节中，我们在交叉检验中讨论了模型选择，其中的一个关键点就是模型的性能评价。本节介绍用于计算分类算法性能的各种评价标准，包括准确率、混淆矩阵、精确率、召回率、F1 度量、ROC 曲线和 AUC 等。同时，我们也会介绍 R 中用来计算这些指标的常用的包和函数。

在分类算法中，我们的预测主要有两种形式：离散型的输出和连续型的输出。离散型的输出就是直接输出分类结果，即每个样本的预测类别。与回归问题类似，分类模型也可以产生连续值的输出。通常连续型输出表现为概率的形式，如每个样本属于每一类的概率。对于实际中的大部分例子，最终的预测类别是必需的。例如，在欺诈检测中，我们希望分类算法能够明确回答每个交易是否是欺诈。

直观地讲，明确的类别输出更易使用。然而，连续型的输出（如概率）能够给我们更多的关于分类模型的信息。还是以欺诈检测为例，假设我们规定模型输出的概率值大于 0.5 则判定为欺诈，否则不为欺诈。考虑两个样本，第一个对应的概率是 0.51，第二个对应的概率是

0.99。根据我们的假设，两个样本都要被判定为欺诈。但是，显然我们对于第二个样本判定为欺诈的信心更高。在一些应用中，分类模型的输出可作为新的特征来训练下一步的模型。这时，连续型的输出能够提供更多有用的信息。在欺诈检测中，利用分类模型输出的概率值，并考虑实际调查该欺诈交易的成本和潜在损失，我们可以决定是否进行下一步的调查。

在下面介绍的算法评价标准中，除了 ROC 曲线和 AUC 需要分类算法的连续型输出，其他指标都只需要直接的分类结果。

6.7.1 准确率

准确率 (accuracy) 是分类问题中最常用的度量。使用准确率时，我们要求算法返回明确的类别。准确率的定义为：

$$\text{准确率} = \frac{\text{正确分类的样本数}}{\text{总样本数}}$$

根据定义，准确率的取值在 0 和 1 之间，越大说明分类结果越好。

但在很多实际应用中，只用准确率一项来衡量分类结果会有失偏颇。如果在分类问题中有一类的样本占大多数，而分类的重点在样本数较少的那一类，那么简单计算准确率并不能较好地反映算法的性能。在前面讨论的欺诈检测的例子中，假设我们有一个包含 100 个样本的数据集，其中 97 个是正常交易，而剩下的 3 个是欺诈交易。如果算法 A 将所有的样本都判定为正常交易，而算法 B 将 8 个交易判定为欺诈交易（包括 3 个真正的欺诈交易）。根据准确率的定义，算法 A 的准确率为 0.97，算法 B 的准确率为 0.95。虽然算法 A 的准确率更高，但是它本质上并没有给我们提供关于欺诈的有效信息。因此，虽然算法 B 的准确率较低，但是我们认为在这个应用场景中算法 B 提供了更多关于欺诈的有效信息，此处算法 B 比算法 A 更适用。

6.7.2 混淆矩阵

在上面的欺诈检测的例子中，其中错误的分类有两种：

- 正常的交易被判定为欺诈；
- 欺诈的交易被判定为正常。

显然，我们对第二种错误更感兴趣，因为它所对应的代价更高，更应该避免。在这类错误代价不同的应用中，简单使用准确率并不能很好地衡量算法的好坏。在这种情况下，我们可以考虑使用混淆矩阵来比较算法 A 和算法 B。

混淆矩阵 (confusion matrix)，又称为可能性表格或错误矩阵。混淆矩阵全面考虑了各种错分的情况，并将分类的结果以“类别 a 被判定为类别 b 的样本数为多少”的形式展示。其中，每一行代表一个实际的类别，每一列代表一个预测的类别。对于最常见的两类分类，混淆矩阵是一个 2×2 的矩阵。如果是 3 类分类问题，则混淆矩阵是一个 3×3 的矩阵。

下面用一个具体的例子来说明如何计算混淆矩阵。在前面的欺诈检测例子中，我们有两个算法。算法 A 将所有的样本都判定为正常的交易，那么它对应的混淆矩阵见表 6-7。

表 6-7 算法 A 对应的混淆矩阵

实际的类别	预测的类别	
	欺诈	正常
欺诈	0	3
正常	0	97

算法 B 将 8 个交易判定为欺诈交易（包括 3 个真正的欺诈交易），对应的混淆矩阵见表 6-8。其中所有的 3 个欺诈交易的样本都被正确判定为欺诈；而正常的 97 个样本中，有 5 个也被判定为欺诈，剩余的 92 个正常样本被判定为正常。

表 6-8 算法 B 对应的混淆矩阵

实际的类别	预测的类别	
	欺诈	正常
欺诈	3	0
正常	5	92

从表 6-7 和表 6-8 的对比可以看出，算法 A 把欺诈判定为正常的有 3 例，而算法 B 为 0。对于这个欺诈检测例子，由于错分所导致的代价不一样，因此相对准确率来说，混淆矩阵是一个更好的评价工具。在实际中，我们可以根据混淆表的每一项对应的权重来选择更优的算法。

在两类分类问题中，2×2 的混淆矩阵的各个元素有专门的名字，见表 6-9。其中 True Positive (TP) 表示实际是正类，同时也被判定为正类；False Positive (FP) 表示实际是负类，但是被判定为正类；False Negative (FN) 表示实际是正类，但是被判定为负类；True Negative (TN) 表示实际是负类，同时也被判定为负类。

表 6-9 TP、FP、TN、FN 的定义

实际的类别	预测的类别	
	正类	负类
正类	True Positive	False Negative (Type II Error)
负类	False Positive (Type I Error)	True Negative

在表 6-9 中，我们把 False Positive 的情况称为第一类错误 (Type I Error)，把 False Negative 的情况称为第二类错误 (Type II Error)。对于 True Positive 和 True Negative，由于我们将其正确分类了，因此就不属于错误之列了。

对于两类分类问题，基于混淆矩阵，还可以定义 True Positive Rate (TPR)，或者称为灵敏度 (sensitivity)。其定义为正类样本中被正确分类的比例：

$$sensitivity = TPR = \frac{TP}{TP + FN} \tag{6-88}$$

类似地，可以定义 True Negative Rate (TNR)，或者称为特异度 (specificity)，其定义为负类样本中被正确分类的比例：

$$specificity = TNR = \frac{TN}{TN + FP} \quad (6-89)$$

还可以定义 False Positive Rate (FPR) 为负类样本中被错误分类的比例:

$$FPR = \frac{FP}{FP + TN} \quad (6-90)$$

定义 False Negative Rate (FNR) 为正类样本中被错误分类的比例:

$$FNR = \frac{FN}{FN + TP} \quad (6-91)$$

利用混淆矩阵, 可以简单地计算准确率 (accuracy):

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (6-92)$$

6.7.3 精确率、召回率和 F1 度量

在很多实际应用中, 我们主要关注正类样本的分类情况。例如, 在前述的欺诈检测例子中, 我们主要关注欺诈的交易, 可以将欺诈的交易定义为正类, 而正常的交易定义为负类。对于这样的问题, 常用的评价标准有精确率 (precision)、召回率 (recall) 和 F1 度量 (F1-Measure), 其具体定义如下:

$$precision = \frac{TP}{TP + FP} \quad (6-93)$$

$$recall = \frac{TP}{TP + FN} \quad (6-94)$$

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2precision \times recall}{precision + recall} = \frac{2TP}{2TP + FP + FN} \quad (6-95)$$

其中 F1 度量是精确率和召回率的调和平均数。事实上, 召回率就是我们前面所讨论的 TPR, 亦等于 $1 - FNR$ 。

精确率度量了在算法判定为正类的所有样本中, 真正的正类样本所占的比例。如果精确率为 1, 说明所有判定为正类的样本都是正类的。精确率越高, 说明判定为正类的样本中事实上为负类的样本越少。而召回率度量了所有的正类样本被正确分类的比例。召回率越高, 说明正类样本被错误分类的比例越小。

但在实际例子中, 很多算法并不能很好地同时优化精确率和召回率。例如, 如果一个算法把所有的样本都判定为正类, 根据精确率和召回率的定义, 我们知道召回率为 1, 但精确率却很低; 另外一个极端的例子是, 如果算法只把那些非常肯定为正类的样本判定为正类, 虽然可以得到很高的精确率, 但是召回率却很低。

因此, 在实际应用中, 需要同时考虑精确率和召回率。F1 度量较好地解决了这个问题。根据

F1 度量的定义,它是精确率和召回率的调和平均数。而调和平均数更接近于精确率和召回率中更小的那个。因此,如果 F1 度量较大,则意味着精确率和召回率都较好,算法的性能较好。

下面我们用一个具体的例子说明如何计算前面讨论过的这些分类算法的评价指标。

例 6-5 我们仍然以欺诈检测中的两个算法为例计算它们的各种指标。考虑算法 B 的分类结果,其混淆矩阵见表 6-8。以上讨论过的分类指标计算如下:

$$TP = 3, FP = 5, FN = 0, TN = 92$$

$$sensitivity = TPR = \frac{TP}{TP + FN} = \frac{3}{3 + 0} = 1.0$$

$$specificity = TNR = \frac{TN}{TN + FP} = \frac{92}{92 + 5} = 0.9484536$$

$$FPR = \frac{FP}{FP + TN} = \frac{5}{5 + 92} = 0.05154639$$

$$FNR = \frac{FN}{FN + TP} = \frac{0}{0 + 3} = 0$$

$$precision = \frac{TP}{TP + FP} = \frac{3}{3 + 5} = 0.375$$

$$recall = \frac{TP}{TP + FN} = \frac{3}{3 + 0} = 1.0$$

$$F1 = \frac{2TP}{2TP + FP + FN} = \frac{2 \times 3}{2 \times 3 + 5 + 0} = 0.5454545$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{3 + 92}{3 + 92 + 5 + 0} = 0.95$$

6.7.4 ROC 曲线和 AUC

前面所讨论的评价标准仅适用于分类器的输出是离散的情况(即输出明确的类别)。当分类器的输出为连续值(如概率值)时,我们需要合适的评价标准。这里我们首先引入接收者操作特征曲线(receiver operating characteristic curve, 简称 ROC 曲线)。

ROC 曲线最早发端于信号处理。在 ROC 曲线中,横坐标是 False Positive Rate (FPR),纵坐标为 True Positive Rate (TPR)。使用 ROC 曲线的前提是分类算法能够输出连续值。在画 ROC 曲线时,我们将分类阈值从大变到小,从将所有的样本都判定为负类到将所有的样本都判定为正类。这样我们可以得到不同的分类结果,对应不同的 FPR 和 TPR 的值,从而得到 ROC 曲线。下面我们通过考虑 ROC 曲线的几个关键点来说明 ROC 曲线的具体画法。图 6-23 就是一个简单的 ROC 曲线。

在 ROC 曲线中,我们要着重考虑以下 4 个点,通过它们可以进一步理解 ROC 曲线。其中第一个点是(0,0),即 FPR=TPR=0,也即 FP=TP=0。此时分类阈值极大,分类器将所有的

样本都判定为负样本了。第二个点是(0,1)，即 $FPR=0$ ， $TPR=1$ ，也即 $FP=FN=0$ 。此时分类器将所有的样本都正确分类了。第三个点是(1,0)，即 $FPR=1$ ， $TPR=0$ ，也即 $TP=TN=0$ 。此时分类器将所有的样本都错误分类了。第四个点是(1,1)，即 $FPR=1$ ， $TPR=1$ ，也即 $TN=FN=0$ 。此时分类阈值极小，分类器将所有的样本都判定为正类了。

因此，ROC 曲线越接近左上角（点(0,1)），分类器的性能越好。虽然 ROC 曲线给出了丰富的信息，但是在很多情况下，我们需要一个类似于准确率的单个值来衡量分类模型的好坏。实际中人们经常使用 ROC 曲线下的面积来衡量算法的好坏，称为 AUC（area under ROC curve）。可以看出 $0 \leq AUC \leq 1$ 。如果分类器 f 的输出对于任意的负类样本 \mathbf{x}_- 和正类样本 \mathbf{x}_+ 满足 $f(\mathbf{x}_-) < f(\mathbf{x}_+)$ ，则该分类器的 ROC 曲线经过点(0,1)，对应的 AUC 值为 1。如果一个分类器随机猜测样本对应的类别，其对应的 AUC 为 0.5。一般来说，一个“正常”分类器的 AUC 介于 0.5~1 之间。如果一个分类器的 AUC 低于 0.5，则意味着它还不如随机猜测。

除了 ROC 曲线下的面积，AUC 还有直观的解释。可以严格地证明 AUC 等于任意选择的一对负类样本 \mathbf{x}_- 和正类样本 \mathbf{x}_+ 满足 $f(\mathbf{x}_-) < f(\mathbf{x}_+)$ 的概率，即

$$AUC = P(f(\mathbf{x}_+) > f(\mathbf{x}_-)) \quad (6-96)$$

因此，AUC 值越大，意味着分类器的性能越好。利用这个公式，可以很容易地理解为什么一个随机猜测的分类器对应的 AUC 为 0.5。在实际中，有些算法会直接利用式(6-96)来优化 AUC 值。

ROC 曲线的另外一个良好的性质是：即便在类不平衡的时候，它都是一个良好的算法评价指标。而在这种情况下，常用的准确率等评价指标容易受到样本数较多的类的影响，不能很好地反映算法实际的好坏。

下面通过一个例子来详细介绍如何画 ROC 曲线和计算对应的 AUC。

例 6-6 假设我们有表 6-10 所示的 8 个样本，其中第二列给出了我们的预测 $P(y=1|\mathbf{x})$ ，已经按 $P(y=1|\mathbf{x})$ 的取值进行了排序；第三列给出了实际类别，其中 1 是正类，-1 是负类。下面我们画出 ROC 曲线并计算对应的 AUC。

表 6-10 诸样本对应的预测 $P(y=1|\mathbf{x})$ 和真实类别

样 本 编 号	$P(y=1 \mathbf{x})$	实 际 类 别
1	0.1	-1
2	0.3	1
3	0.4	-1
4	0.5	1
5	0.55	1
6	0.7	-1
7	0.8	1
8	0.85	1

为了画出 ROC 曲线，我们需要改变分类阈值得到不同的 FPR 和 TPR 的值。开始时我们

假定所有的样本都被判定为负类，这样可得 $FPR=TPR=0$ ，对应的 TP、TN、FP、FN、FPR 和 TPR 等具体计算结果见表 6-11 的“正类数目为 0”行。接下来我们改变分类阈值，使得 8 个样本中 7 个被判定为负类，而只有编号为 8 的样本被判定为正类。相应地，我们可以再次计算 FPR 和 TPR 等值，参见表 6-11 的“正类数目为 1”行。随着分类阈值的不断增加，最终所有的样本都被判定为正类，此时有 $FPR=TPR=1$ 。

表 6-11 利用排序更有效地计算 FPR 和 TPR

正类数目	TP	TN	FP	FN	FPR	TPR
0	0	3	0	5	0	0
1	1	3	0	4	0	0.2
2	2	3	0	3	0	0.4
3	2	2	1	3	0.3333	0.4
4	3	2	1	2	0.3333	0.6
5	4	2	1	1	0.3333	0.8
6	4	1	2	1	0.6667	0.8
7	5	1	2	0	0.6667	1
8	5	0	3	0	1	1

利用表 6-11 中的 FPR 和 TPR 的值，可以得到如图 6-23 所示的 ROC 曲线，同时也可以得到 $AUC=0.7333$ 。

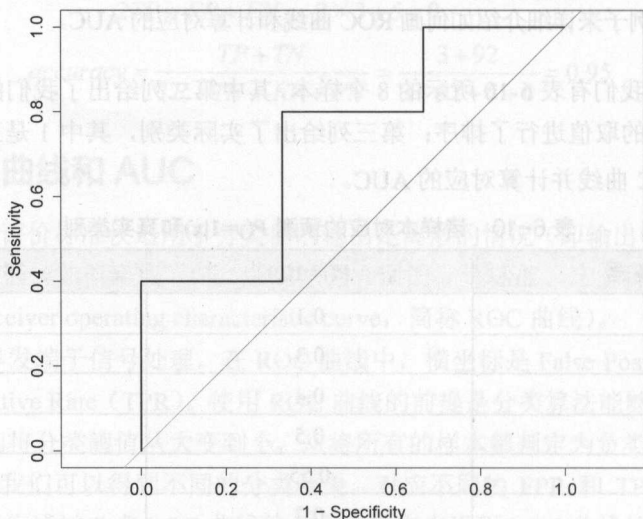


图 6-23 示例数据所对应的 ROC 曲线

6.7.5 R 中评价标准的计算

本节主要介绍如何使用 R 中的包和相关函数来计算上面介绍的分类算法常用评价标准。具体的 R 代码参见文件 `classification_performance_metric.R`。

这里我们仍使用表 6-10 中的数据。该数据集包括 8 个样本，在代码中用 `label_list` 表示其真实的类别，`prob_list` 表示分类算法输出的概率。我们规定概率大于或等于 0.5 的样本判定为正类，并将结果保存在 `pred_list` 中。注意，这里我们将 `label_list` 和 `pred_list` 显式地转换为因子类型。

```
prob_list = c(0.1, 0.3, 0.4, 0.5, 0.55, 0.7, 0.8, 0.85)
pred_list = ifelse(prob_list >= 0.5, 1, -1)
label_list = c(-1, 1, -1, 1, 1, -1, 1, 1)
# In R, we use factor to store the label list
pred_list <- as.factor(pred_list)
label_list <- as.factor(label_list)
```

首先计算准确率并打印出来：

```
# Part 1. Compute accuracy
accuracy = sum(pred_list == label_list) / length(label_list)
msg = paste0('accuracy = ', accuracy)
print(msg)
```

其输出如下：

```
"accuracy = 0.75"
```

然后调用 `caret` 包中提供的 3 个函数来计算混淆矩阵、灵敏度和特异度：

```
library('caret')
confM1 <- confusionMatrix(data=pred_list, reference=label_list, positive='1')
print('the confusion matrix and other metrics are')
print(confM1)

sens <- sensitivity(data=pred_list, reference=label_list, positive='1')
spec <- specificity(data=pred_list, reference=label_list, negative='-1')
msg <- paste0('sensitivity = ', sens, ' and specificity = ', spec)
print(msg)
```

上述代码使用 `caret` 包中的 `confusionMatrix` 函数计算混淆矩阵，使用 `sensitivity` 函数计算灵敏度，使用 `specificity` 函数计算特异度。在使用这 3 个函数时，`data` 参数对应我们预测的类别，而 `reference` 对应真正的类别；我们可以使用参数 `positive` 或者 `negative` 来分别指定对应的正类或者负类。下面是对应的输出：

```
[1] "the confusion matrix and other metrics are"
Confusion Matrix and Statistics
```



```

Reference
Prediction -1 1
           -1 2 1
           1 1 4

Accuracy : 0.75
95% CI : (0.3491, 0.9681)
No Information Rate : 0.625
P-Value [Acc > NIR] : 0.3697

```

```

Kappa : 0.4667
McNemar's Test P-Value : 1.0000

```

```

Sensitivity : 0.8000
Specificity : 0.6667
Pos Pred Value : 0.8000
Neg Pred Value : 0.6667
Prevalence : 0.6250
Detection Rate : 0.5000
Detection Prevalence : 0.6250
Balanced Accuracy : 0.7333

```

```
'Positive' Class : 1
```

```
[1] "sensitivity = 0.8 and specificity = 0.6666666666666667"
```

可以看出,调用 `confusionMatrix` 函数时,它除了计算混淆矩阵外,还计算了准确率、灵敏度和特异度等很多相关的评价指标。而 `sensitivity` 函数和 `specificity` 函数仅仅计算了相关的灵敏度和特异度。

接下来我们使用 `pROC` 包中的 `roc` 和 `auc` 函数分别画出 ROC 曲线和计算对应的 AUC 值。在前面的讨论中,我们介绍了理论上画 ROC 曲线和计算 AUC 的方法。在实际中,我们推荐读者直接使用 `pROC` 包中的对应函数。

```

library('pROC')
# create an roc object using the roc function in pROC
rocCurve <- roc(response=label_list,
                 predictor=prob_list)
# compute the AUC using the auc function
auc_value = auc(rocCurve)
print(auc_value)
# plot the ROC curve
plot(rocCurve, legacy.axes=T)

```

在使用 `pROC` 包中的 `roc` 函数时,首先使用 `roc` 函数创建一个 `roc` 对象。在使用 `roc` 函数时, `response` 参数表示真正的类别,而 `predictor` 参数表示我们的预测值。得到 `roc` 对象后,可使用 `auc` 函数计算 ROC 曲线对应的 AUC 值,也可使用 `plot` 函数画出对应的 ROC

曲线。print(auc_value) 的输出如下：

```
Area under the curve: 0.7333
```

使用 plot 函数生成的 ROC 曲线如图 6-23 所示。

表 6-12 总结了在 R 中计算这些常用评价标准的包和函数。

表 6-12 R 中常用的计算分类算法标准的包和函数

评价标准	R 包	函 数
混淆矩阵	caret	confusionMatrix
AUC	pROC	roc 和 auc
ROC	pROC	roc
灵敏度	caret	sensitivity
特异度	caret	specificity

6.8 不平衡分类问题

本节讨论不平衡分类 (imbalanced classification) 问题。以两类分类问题为例，在不平衡分类问题中，训练集中的样本主要属于某一类，而另一类的样本很少，使得分类器难以学到关于较少样本对应的类的特征。在实际中，如果我们对样本更少的那类更感兴趣，简单使用前面讨论的分类算法很难得到“好”的分类器。我们前面讨论的欺诈检测就是这样一个例子。在欺诈检测中，大部分交易都是正常的，真正欺诈的交易太少，而我们恰恰对欺诈的交易最感兴趣。

不失一般性，在不平衡问题中，我们假设正类的样本很少而负类的样本很多。因此，正类样本的一些“模式”淹没在负类样本的“模式”中了，导致在绝大多数情况下分类器都很难直接对正类样本做出较好的处理。本节详细讨论处理不平衡分类的有效方法，主要包括优化不同的算法评价标准、改变样本权值、通过取样改变训练集分布和代价敏感学习 (cost-sensitive learning)。

6.8.1 使用不同的算法评价标准

在前面的欺诈检测例子中，由于正类样本的数目太少，同时用户对于正类样本的正确分类更为关注，因此准确率在该应用中并不是一个良好的算法评价标准。

一种较简单的解决方案是选择最符合实际的算法评价标准来选择最优的参数值。例如，在欺诈检测的例子中，我们可以最大化 AUC，这也是我们提出不同算法评价标准的初衷之一。

6.8.2 样本权值

很多分类算法都允许给每个训练样本赋予不同的权值。这样，对于样本数目较少的类别，我们可以给其对应的样本赋予更大的权值，从而使得算法能够发现可以对它们进行准确分类

的“模式”。赋予样本不同权值的方法在集成学习的 boosting 中应用很广。但该方法的缺陷是需要分类算法支持为每个训练样本设置不同的权值。如果分类算法本身不支持样本权重，这种方法就不好直接应用。

6.8.3 取样方法

取样是一种广泛用来解决不平衡分类问题的方法。取样方法的优点是只需要通过取样改变训练集中正负类样本的分布，使得样本的分布从不平衡到平衡，而分类算法不需要做任何改变就可以直接应用了。事实上，取样方法可以认为是 6.8.2 节样本权值方法中将权重设为整数的特殊情况。举个简单的例子，假设我们在新的训练集中有两个样本 x_i ，等价于赋予样本 x_i 权值 2。

在不平衡分类问题中，主要有两种取样方法：下取样（down-sampling）和上取样（up-sampling，也称为 oversampling）。其中，下取样就是在样本多的类中取样，使得该类的样本数降低；上取样则相反，通过模拟或者直接复制正类样本点，使得正类样本点的样本数增加。此外，还有下取样和上取样同时使用的混合型取样方法。

下面我们仍以欺诈检测为例来介绍下取样和上取样的具体方法。在欺诈检测的例子中，假设原始的训练集中有 50 个正类样本（欺诈样本）和 5000 个负类样本（正常样本）。

在一个简单的上取样方法中，我们可以把每个正类样本复制 99 次从而得到 5000 个正类样本，然后和原来的 5000 个负类样本一起组成新的训练集。在这个新的训练集中，没有类不平衡的问题。该方法简单，但是如果正类样本中有噪声的话，则噪声也被放大了若干倍，从而影响分类器的最终性能。

在一个简单的下取样方法中，我们随机地从 5000 个负类样本中取出 50 个负类样本，然后和原来的 50 个正类样本一起组成新的训练集。但是由于我们损失了很多负类样本的信息，利用这个新训练集学习得到的分类器有可能遗失了负类样本的一些重要信息。

在实际中，有很多改进措施可以提高下取样的性能。例如，可以在取样时偏向那些离分类界限较近的负类样本而忽视那些离分类界限较远的负类样本。更理想的做法是引入集成学习的一些做法。例如，可以使用前面讨论的简单下取样方法 k 次，并且在每个训练集上训练一个分类器。这样就得到了 k 个分类器，最后的分类器是所有 k 个分类器的平均。这个方法考虑了更多的负类样本的信息，通常可以取得较好的分类性能，缺点是计算复杂度偏大。更复杂的方法包括：在每次取样时使用 bootstrap 取样（可重复取样，将在第 9 章介绍），并保证每类所取样的样本数相等。和前一种方法类似，对于每个取样得到的训练集，我们建立一个分类器，而最终的分类器是所得的多个分类器的平均。这样既解决了不平衡分类的问题，同时又在建立多个分类器时引入了多样性，从而使得最终的分类器性能较好。实质上，该方法类似于集成学习的思想，而随机森林（见第 9 章）的一种具体实现就和我们这里描述的下取样方法非常相似。

总之，取样方法的优点是取样后所有的分类算法都可直接应用；缺点是在取样过程中，会遗失有效信息（下取样）或者添加一些噪声（上取样）。

6.8.4 代价敏感学习

我们知道，在机器学习中，很多分类算法都是通过最小化损失函数得到相应的模型。代价敏感学习（cost-sensitive learning）的核心思想是在分类算法所考虑的损失函数中，为正类（样本数较少）的样本错分所导致的错误赋予较大的权值，从而能够在学习模型的过程中对正类样本给予更多的关注。例如，在欺诈检测的例子中，将正类（欺诈）样本错误分类成负类（正常交易）的代价远高于将负类样本错误分类为正类样本的代价。通过直接改变损失函数，我们能够得到更好的处理不平衡分类的分类器。与前面介绍的方法相比，代价敏感学习要求对分类算法本身进行修改。

在两类分类问题中，我们考虑给不同的分类情况赋予不同的权重，见表 6-13。其中，将正类分为正类的权重为 c_{11} ，将正类分为负类的权重为 c_{10} ，将负类分为正类的权重为 c_{01} ，将负类分为负类的权重为 c_{00} 。

表 6-13 不同分类给予的不同权重

实际的类别	预测的类别	
	Positive Class (正类)	Negative Class (负类)
Positive Class (正类)	c_{11}	c_{10}
Negative Class (负类)	c_{01}	c_{00}

在损失函数中，我们通常只考虑错误分类的情况，包括：（1）将正类错分为负类；（2）将负类错分为正类。换言之， $c_{11} = c_{00} = 0$ 。考虑不同的权重 c_{10} 和 c_{01} 后，损失函数可以表示为

$$L = c_{10} \sum_{x_i \in C_{10}} L(y_i, f(x_i)) + c_{01} \sum_{x_i \in C_{01}} L(y_i, f(x_i)) \quad (6-97)$$

这里我们使用 C_{10} 、 C_{01} 分别表示在分类中被错分为负类和正类的样本集合。如果使用最简单的 0-1 损失函数，我们有

$$L = c_{10}FN + c_{01}FP \quad (6-98)$$

下面以支持向量机和决策树作为两个实际的例子来进一步说明如何进行代价敏感学习。

对于支持向量机而言，在损失函数中我们可以为每一类赋予不同的权重。在标准的支持向量机中，对于两类分类问题我们考虑如下的损失函数（包括正则化项）：

$$\min_{w, b} \frac{1}{2} \|w\|_2^2 + c \sum_{i=1}^n \varepsilon_i \quad (6-99)$$

这里 $\varepsilon_i = \max\{0, 1 - y_i(w^T x_i + b)\}$ ，表示样本 x_i 被错分时的损失。在标准的支持向量机中，我们对所有的样本都一视同仁。在上面的公式中，我们给所有样本的错分都赋予了权重 c 。在不平衡分类中，我们可以赋予不同的权重，从而得到新的损失函数（包括正则化项）：

$$\min_{w, b} \frac{1}{2} \|w\|_2^2 + c_1 \sum_{x_i \in C_1} \varepsilon_i + c_2 \sum_{x_i \in C_2} \varepsilon_i \quad (6-100)$$

这里我们把正类和负类样本的错分权重分别设为 c_1 和 c_2 , 被错分为负类和正类的样本集合分别为 C_1 和 C_2 。例如, 在欺诈检测的例子中, 可以令 $c_1 = 100, c_2 = 1$ 来着重强调正类的正确分类。

在构建决策树的过程中, 通过为不同的类引入不同的代价, 我们可以改变构建决策树的过程, 例如, 在每一步如何选择特征进行分割、如何决定叶结点的类别等。下面我们简单介绍一下如何在引入不同代价时决定叶结点的类别。

在决策树中, 我们通常直接按照少数服从多数的原则来决定叶结点对应的类别。换言之, 训练集中属于该结点的样本中最主要的类别就是整个叶结点对应的类别。但是在代价敏感学习中, 由于我们给不同的错误引入了不同的权重, 所以需要引入新的规则。在两类问题中, 假设我们仍旧考虑最简单的 0-1 损失函数, 并假设某叶结点对应的正类样本数目为 N_+ , 负类样本数目为 N_- 。根据前面的讨论, 损失函数为

$$L = c_{10}FN + c_{01}FP \quad (6-101)$$

当把所有结点都判定为正类时, 我们有

$$TP = N_+, FP = N_-, FN = 0, TN = 0 \quad (6-102)$$

对应的损失函数 L^+ 为:

$$L^+ = c_{10}FN + c_{01}FP = c_{01}N_- \quad (6-103)$$

当把所有结点都判定为负类时, 我们有:

$$TP = 0, FP = 0, FN = N_+, TN = N_- \quad (6-104)$$

对应的损失函数 L^- 为:

$$L^- = c_{10}FN + c_{01}FP = c_{10}N_+ \quad (6-105)$$

当 $L^+ < L^-$ 时, 我们把该叶结点判定为正类, 因为此时对应的损失函数 L^+ 更小, 对应于如下条件:

$$L^+ < L^- \Leftrightarrow c_{01}N_- < c_{10}N_+ \Leftrightarrow \frac{N_-}{N_+} < \frac{c_{10}}{c_{01}} \quad (6-106)$$

换言之, 当 $\frac{N_-}{N_+} < \frac{c_{10}}{c_{01}}$ 时, 该叶结点应判定为正类; 否则判定为负类。当 $c_{10} = c_{01}$ 时, 该规则为

$$\frac{N_-}{N_+} < 1 \quad (6-107)$$

这就是我们前面讨论的少数服从多数原则。在欺诈检测中, 我们令 $c_{10} = 100, c_{01} = 1$ 以突出欺诈的样本, 则规则为

$$\frac{N_-}{N_+} < 100 \quad (6-108)$$

在新的规则下, 就算是负类 (正常样本) 更多, 但是只要负类与正类样本数的比值低于 100, 我们仍然将其判定为正类 (欺诈)。

第 7 章 推荐算法

在信息时代，大多数人的生活中都会接触超量的信息。为了帮助用户从海量的信息中提取有用的信息，推荐系统（recommendation system）应运而生。目前，推荐系统已经广泛应用于各个领域，最典型的例子是电子商务领域。例如，在淘宝网或者 Amazon.com 上，根据每个用户过去的购买记录，网站可以推测该用户会对哪些产品感兴趣并进行推荐以帮助用户购买商品。在视频推荐领域，Netflix 公司在 2006 年举办了关于电影推荐的 Netflix Prize^① 比赛并取得了极大的成功，有力地推动了推荐算法的研究。目前微软^② 和谷歌^③ 等公司都推出了各自的通用推荐系统引擎。

在本章，我们将介绍推荐算法的原理，并讨论不同类型的推荐算法，包括基于内容的推荐算法和基于协同过滤（collaborative filtering）的推荐算法。由于协同过滤推荐算法在工业界取得的巨大成功，因此我们主要介绍该类算法，包括基于矩阵分解和基于邻域的推荐算法。本章介绍的算法都是较为实用的基本推荐算法。在实际使用推荐算法时，用户应该结合数据本身的特性选择合适的算法。

在介绍具体的推荐算法之前，首先讨论推荐系统基础，包括推荐系统中的一些基本概念、推荐算法的评价标准等。此外，为了方便读者使用常用的推荐算法，还将介绍如何使用 R 中的 recommenderlab 软件包。

7.1 推荐系统基础

推荐系统的主要任务是根据用户、商品的历史信息，尤其是用户和商品的交互信息，为每个用户推荐他最喜欢的商品。

在推荐算法中，有两类基本对象：

- (1) 用户（user）；
- (2) 商品（item）。

事实上，商品是我们在推荐系统中所有待推荐对象的统称，在英文文献关于推荐系统的讨论中一般称为 item。根据实际问题，商品的具体定义会有相应的变化。例如，在淘宝网或者 Amazon.com 上，每件商品（item）就是一件待售的商品。在视频网站（如优酷）上，商品

① <http://www.netflixprize.com/>

② <https://azure.microsoft.com/en-us/documentation/articles/machine-learning-recommendation-api-documentation/>

③ <https://cloud.google.com/solutions/recommendations-using-machine-learning-on-compute-engine>

就是一段视频。

在推荐算法中，一个核心问题是：根据用户和商品的历史信息，如何估计用户 u 对某件商品 i 的喜好程度。根据估计的喜好程度，系统可以向用户推荐更加新颖和有趣的商品。因此，在推荐系统中的一个核心概念是用户对于商品的喜好程度，这也是我们建模的核心。

在实际的推荐问题中，我们可以利用的数据主要是用户以往的购买历史，以及用户以往对于商品的评价数据。例如，用户 u 在 Amazon.com 购物后对所购商品很满意，给了一个五星的评价。除了直接的评价数据，很多时候我们还可以利用用户的间接评价数据。例如，用户浏览某在线商店网站时，虽然没有购买某件商品，但是浏览过，显示用户对该商品感兴趣。

除了用户-商品的交互信息外，其他可以利用的数据包括：

- (1) 商品的信息，包括商品的价格、类型；
- (2) 用户的信息，如用户的性别、年龄、居住地点等。

下面我们讨论推荐算法的一些基础知识。根据算法的实现原理，推荐算法可以分为两大类：

- (1) 基于内容的推荐算法；
- (2) 基于协同过滤的推荐算法。

基于内容的推荐算法的核心思想是利用商品的相关信息为每件商品构建一个特征向量来表示对应的商品。例如，淘宝网上每件商品的价格、用途、生产厂家等。在此基础上，根据用户过去购买和评价相关商品的历史数据，可以为每个用户构建相应的特征向量以代表该用户。这样就能通过计算用户和商品之间的相似度来预测用户对于商品的喜好程度。

与基于内容的推荐算法相比，基于协同过滤的推荐算法的核心思想是利用用户和商品之间的相互关系来构建推荐模型和进行推荐。所谓协同过滤，就是找出类似的用户或者商品（这一阶段可以称为“协同”），再利用类似用户的喜好或者类似商品所受到的评价情况，来预测用户对商品的喜好程度并挑出最喜欢的商品（这一阶段可以称为“过滤”）。在目前的工业界实践中，基于协同过滤的推荐算法取得了极大的成功和广泛的应用。

在协同过滤中，我们用 r_{ui} 表示用户 u 和商品 i 的关系。例如，在网络购物网站中，可以利用用户-商品的购买信息来分析两者之间的关系。这里的用户-商品的关系可以表示为：

- $r_{ui} = 1$ ，如果用户 u 购买过商品 i ；
- $r_{ui} = 0$ ，如果用户 u 没有购买过商品 i 。

用户-商品信息的其他例子包括用户对于商品的评价信息（如 1 星~5 星）、用户对于商品的浏览信息等。根据实际问题的不同，用户和商品的关系 r_{ui} 的形式通常可以分为 4 类。

(1) 标量型 (scalar) 或者数值型 (numeric)。例如，用户对于电影的评价可以使用评分 1~5 来评价，分数越高，评价越好。

(2) 排序型 (ordinal)。一个典型例子是问卷调查中的答案，如强烈同意/同意/中立/不同意/强烈不同意等。

(3) 布尔型 (boolean)：用户对于商品的评价只有两个值，如喜欢/不喜欢。

(4) 一元值型 (unary)：用户购买了某件商品，或者用户浏览了某段视频。

我们可以将所有的 r_{ui} 放在一个矩阵中，其中每一行对应一个用户，每一列对应一件商品。

一般来说, 协同过滤算法主要利用该矩阵进行建模和推荐。当然, 在实际中, 如果还有其他可用的信息, 也可以在算法中使用。在下面的讨论中, 我们将 r_{ui} 一律称为用户 u 对于商品 i 的评价 (rating)。事实上, r_{ui} 描述的是用户 u 和商品 i 的相互关系, 对于不同的应用有不同的含义。但是, 我们讨论的算法可以简单地移植到相应的情况中。

协同过滤算法可以进一步分为以下两类算法:

- (1) 基于矩阵分解的推荐算法 (recommendation based on matrix factorization);
- (2) 基于邻域的推荐算法 (neighborhood-based recommendation)。

基于矩阵分解的算法将用户和商品都用一个低维空间的向量来表示, 然后利用该低维表示来进行推荐。从 2006 年开始进行的 Netflix Prize 比赛引起了全世界顶级推荐算法高手的广泛兴趣, 最后获胜的算法就是基于矩阵分解的协同过滤算法。

基于邻域的推荐算法主要考虑商品之间的关系, 或者用户之间的关系。基于邻域的推荐算法的基本思想: 当我们要向用户 u 推荐商品 i 时, 找出与用户 u 相似的用户, 或者与商品 i 相似的商品, 从相似的用户或者商品的历史数据中推断用户 u 对商品 i 的喜好程度。基于邻域的推荐算法可以进一步分为基于商品的邻域推荐算法和基于用户的邻域推荐算法。

在本章中, 我们着重介绍基于协同过滤的推荐算法, 即基于矩阵分解的推荐算法和基于邻域的推荐算法。

在介绍具体的推荐算法之前, 我们讨论评价信息中商品的长尾分布 (long-tail distribution)。在很多实际的推荐问题中, 在所得的评价信息 r_{ui} 中, 一小部分商品出现的频率很高, 而大部分商品的出现频率很低。换言之, 在我们收集到的 r_{ui} 里面, 大部分是关于很少一部分商品的; 而对于绝大多数商品, 只有很少的评价信息。图 7-1 给出了一个实际推荐问题评价数据中商品出现的次数的分布。在图 7-1 中, x 轴对应的是商品按照在评价信息中的流行程度从高到低排序后的编号, y 轴是商品在评价信息中出现的次数。从图 7-1 中可以明显看出, 这里商品出现次数的分布是一个长尾分布。

长尾分布的存在对于推荐问题来说是一个挑战。在推荐算法中, 推荐那些流行的商品是一个比较安全的选择: 用户对于这些推荐的流行商品不会觉得是完全无关的商品。但是在实际中, 那些很流行的产品对于商家来说基本上都是走量的商品, 利润通常都比较低; 反而是那些评价很少的商品利润更高一些。因此, 在实际部署推荐系统时, 商家对于推荐那些不流行的商品的热情更高。一个典型的例子就是 Amazon.com 从销售那些 (总体而言) 不那么流行的书籍中获利颇丰。另一方面, 对于用户来说, 推荐系统推荐的都是那些非常流行的商品, 也会有些枯燥。如果推荐系统能从那些不是那么流行的商品中推荐对用户来说有意思的商品, 对于用户也是具有吸引力的。但是我们也要注意, 在推荐那些不那么流行的商品的过程中, 推荐系统更容易犯错, 因为我们对于流行商品有着充足的信息, 那些不那么流行的商品所拥有的信息则少很多 (从图 7-1 可以直观地看出)。

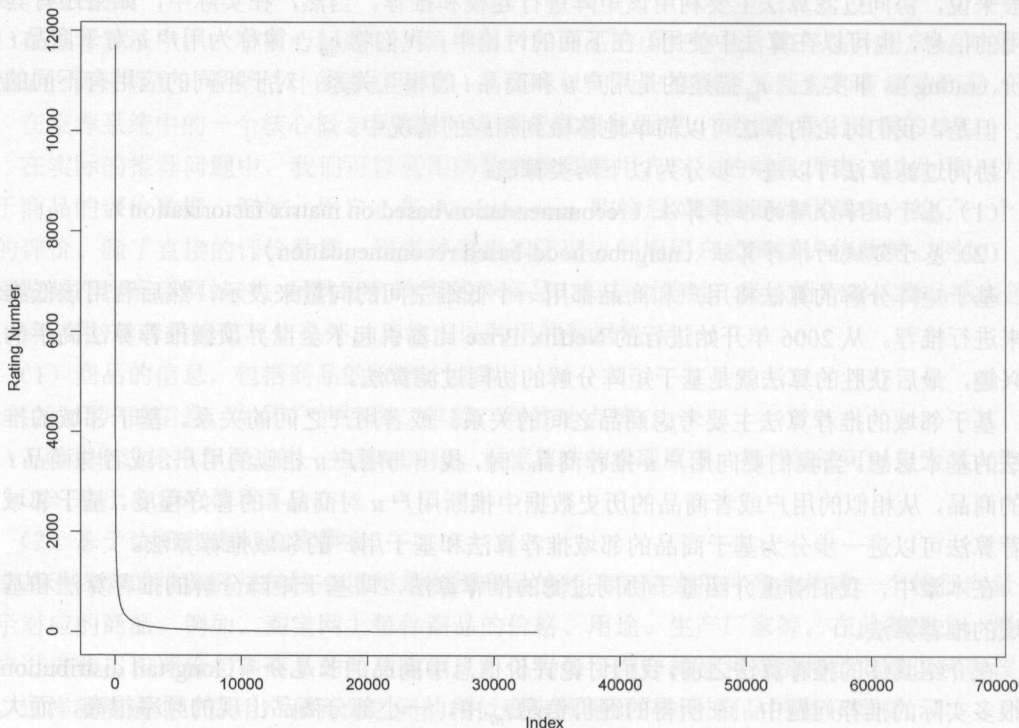


图 7-1 评价数据中商品出现次数的长尾分布示例

7.1.1 常用符号

本章中，用户集记为 U ，商品集记为 I 。用户的总数记为 m ，商品的总数记为 n ，即 $m = |U|, n = |I|$ 。特别地，我们使用 u, v 来作为用户的下标，而使用 i, j, l 作为商品的下标。

我们假设每个用户对于一件商品只有一个评价值，并把用户 $u \in U$ 对于商品 $i \in I$ 的评价记为 r_{ui} 。所有用户对商品的评价值的集合记为 S 。例如，在数值型评分中，很多时候用户用 1 星~5 星来评价，则 $S = \{1, 2, 3, 4, 5\}$ ；在布尔型评分中 S 的取值可以为 $S = \{\text{喜欢}, \text{不喜欢}\}$ 。我们把 r_{ui} 相应的预测值记为 \hat{r}_{ui} 。如果我们将推荐模型记为 f 的话，可以将 \hat{r}_{ui} 记为 $f(u, i)$ 。所有用户-商品对已知的集合记为 $K = \{(u, i) | r_{ui} \text{ 已知}\}$ 。

我们将所有的 r_{ui} 放在一个矩阵中，其中每一行对应一个用户，每一列对应一件商品。我们称这个矩阵为评价矩阵 (rating matrix)，并记为 R 。

对于用户 u ，我们把他/她所评价过的商品集记为 $I(u)$ 。对于商品 i ，我们把所有评价过它的用户集记为 $U(i)$ 。我们把同时被用户 u, v 评价过的商品集记为 $I(u) \cap I(v)$ 。类似地，我们把同时评价过商品 i, j 的用户集记为 $U(i) \cap U(j)$ 。

7.1.2 推荐算法的评价标准

在计算推荐算法的评价标准时，假设我们有一个单独的测试集记为 T 。

在评价推荐算法时，我们可以使用与回归问题类似的指标。通常，我们可以使用平均绝对误差（mean absolute error, MAE）和均方根误差（root mean squared error, RMSE）。对于模型 f ，我们记 $\hat{r}_{ui} = f(u, i)$ 是模型 f 对于用户 u 对商品 i 的评价的预测，则其对应的平均绝对误差和均方根误差定义为：

$$MAE(f) = \frac{1}{|T|} \sum_{r_{ui} \in T} |f(u, i) - r_{ui}| \quad (7-1)$$

$$RMSE(f) = \sqrt{\frac{1}{|T|} \sum_{r_{ui} \in T} (f(u, i) - r_{ui})^2} \quad (7-2)$$

在推荐系统中，通常最重视最前面的推荐项。因此，更常用的一种考察推荐算法性能的方法是考察算法返回的前 k 个推荐项。一般情况下，可以计算精确率（precision）和召回率（recall）。特别是当 r_{ui} 为 0 或者 1 的情况下，这些评价标准特别有效。例如，在有些应用中， $r_{ui} = 1$ 表示用户购买了商品，为 0 则表示没有购买。在下面的讨论中，使用这个购买的例子来说明精确率和召回率的计算和含义。假设对于用户 u ，我们推荐了 k 件商品，并按照（模型预测的）用户 u 对于商品的喜好程度 \hat{r}_{ui} 从高到低排列，把排好的序列记为 $L(u)$ 。在测试集 T 中，把用户 u 购买的商品记为 $I_{test}(u)$ ，则可以定义精确率和召回率：

$$precision(L) = \frac{1}{|U|} \sum_{u \in U} \frac{|L(u) \cap I_{test}(u)|}{|L(u)|} \quad (7-3)$$

$$recall(L) = \frac{1}{|U|} \sum_{u \in U} \frac{|L(u) \cap I_{test}(u)|}{|I_{test}(u)|} \quad (7-4)$$

其中 $precision(L)$ 是精确率，而 $recall(L)$ 是召回率。这里我们考虑了所有的用户的精确率和召回率，并计算平均值作为最后的结果。

一般来讲，精确率较高，表示返回的结果 L 中很多都是用户真正购买的商品。而召回率则反映了我们能够在多大程度上预测用户所购买的所有商品。

在推荐系统中，用户一般对最前面的结果最感兴趣。越是排在前面的，重要性越高。而在精确率和召回率中，我们把所有在返回序列的商品都一视同仁（当然，可以取不同的 k 值来计算精确率和召回率）。在返回的序列 L 中，可以进一步给每个返回商品根据其在 L 中的位置赋予不同的权重，位置越高，权重越大。一种采用这种策略的评价标准称为平均反击中率（average reciprocal hit-rank, ARHR），其定义如下：

$$ARHR(L) = \frac{1}{|U|} \sum_{u \in U} \sum_{i \in I(u)} \frac{1}{rank(i, L(u))} \quad (7-5)$$

这里 $rank(i, L(u))$ 是商品 i 在序列 $L(u)$ 中的排序编号。如果在最前边, $rank(i, L(u)) = 1$; 如果排在第二, 则 $rank(i, L(u)) = 2$; 如果不在序列 $L(u)$ 中, 则 $rank(i, L(u)) = +\infty$ 。

此外, 还可以使用排序算法中的一些指标来度量推荐算法的好坏, 如 DCG 和 NDCG 等。读者可以参阅排序算法中的对应章节。

7.2 基于内容的推荐算法

本章的重点在于协同过滤, 包括基于矩阵的算法和基于邻域的算法。因此, 在本节我们简单介绍基于内容的推荐算法 (content-based recommendation), 包括其基本原理及优缺点。

在基于内容的推荐中, 首先为每件商品 i 构建一个向量 \mathbf{x}_i 来表示该商品的特征。例如, 在新闻推荐中, 为每篇新闻我们可以使用 TF-IDF (Term Frequency-Inverse Document Frequency, 词频-逆文档频率) 来表示。另一个例子是在网络购物网站中, 对每件商品可以使用商品的名称、类别、价格等来构建向量 \mathbf{x}_i 。

在构建完每件商品的特征后, 可以根据商品的特征和用户-商品的交互关系为每个用户构建相应的特征。对于用户 u , 考虑所有在集合 $I(u)$ 中的商品。一种简单的方式是将用户 u 的特征表示为 $I(u)$ 中所有商品 i 的加权和, 且权重为 r_{ui} :

$$\mathbf{x}_u = \sum_{i \in I(u)} r_{ui} \mathbf{x}_i \quad (7-6)$$

在得到商品和用户的特征后, 就可以计算它们之间的相似度。例如, 可以使用余弦相似度来计算 \mathbf{x}_i 和 \mathbf{x}_u 的相似度:

$$s_{ui} = \frac{\mathbf{x}_i^T \mathbf{x}_u}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_u\|_2} \quad (7-7)$$

这样, 对于用户 u , 推荐相似度最高的商品即可。

基于内容的推荐算法依赖于我们对于商品的了解。如果我们对商品没有足够的了解, 就无法为每件商品构建准确的特征以进行合适的推荐。事实上, 很多时候商品已有的信息并不足以构建有效的特征, 因此无法直接使用基于内容的推荐算法。

完全基于商品的特征可能在实际中会导致较差的推荐结果。例如, 两本同样关于 C++ 语言的书, 使用的词汇基本相同, 使得这两本书的特征很相似。但是一本很畅销, 另一本的销售却很差。如果完全使用基于内容的推荐算法, 系统就会把两本书同时向相应的用户推荐。此外, 基于内容的推荐算法永远只能给用户推荐他/她已经熟悉的商品, 而很多与他/她已经熟悉的商品不同的商品则基本不会推荐。而在实际中, 人们往往愿意尝试一些新的商品或者非常流行的商品。从商家的角度讲, 也希望给用户推荐一些新的有趣的商品以提高销售业绩。

在实际部署推荐系统时,一般来说,如果没有足够多的用户-商品交互数据,那么基于内容的推荐算法是一个较好的选择,它能够较好地解决冷启动(cold start)的问题。但是,一旦我们有了足够多的历史数据,协同过滤一般能够取得更高的性能。当然,如果可以的话,同时部署不同类型的推荐算法再综合一般能够得到更为准确的推荐结果。

7.3 基于矩阵分解的算法

由于在 Netflix Prize 中的巨大成功,基于矩阵分解的推荐算法在实际中取得了广泛的应用。在本节中,我们由易到难,详细讨论常用的多种矩阵分解算法,包括:

- 无矩阵分解的基准方法;
- 基于奇异值分解(SVD)的推荐算法;
- 基于 SVD 推荐算法的变体
 - AFM 模型;
 - 翻转的 AFM 模型;
 - ASVD 模型(或者 SVD++模型);
 - 翻转的 ASVD 模型;
 - 引入时间信息的模型。

对于每种矩阵分解算法,首先介绍其基本原理,然后介绍其对应的求解算法。对于矩阵分解算法,通常采用随机梯度下降(stochastic gradient descent)算法来求解模型对应的参数。

基于矩阵分解的推荐算法的基本假设:每个用户和每件商品都可以使用低维向量来表示。与基于内容的算法不同,在基于矩阵分解的算法中,这些低维向量是从用户-商品评价矩阵中“学习”出来的,而在基于内容的推荐算法中,用户和商品的特征向量一般都是从商品的特征中提取出来的。而在这个学习过程中,我们希望这些低维表示能够帮助我们建模用户和商品之间已知的相互关系,进而能够准确地预测未知的 r_{ui} 的值。在本章接下来的讨论中,我们假设 r_{ui} 越高,用户 u 对商品 i 的兴趣越大。

7.3.1 无矩阵分解的基准方法

在推荐问题中,我们要考虑用户和商品之间的关系。在讨论复杂的矩阵分解算法之前,首先介绍基准算法(baseline algorithm)。

在很多实际应用中,虽然用户-商品之间的相关关系很重要,但是很多时候用户对于某件商品的评价首先是受到用户自身或者商品本身的性质决定的。在实际中,有的用户倾向于给更高的评价,而有的用户则一般给出较低的评价。举个简单的例子,用户 u 在某在线商店购物时一般都是给出3星或者以上的评价(可选的评价是1星~5星),而用户 v 则很少给出高于3星的评价,一般都是2星左右。类似地,如果一件商品非常畅销,则其收到的评价一般较高;如果一件商品存在一些问题,则很有可能所收到的评价一般都较低。

在基准算法中,我们暂时不考虑用户和商品之间的相互关系。我们只考虑每个用户总体的偏好和每件商品的总体评价。具体来说,我们可以用如下公式来表示:

$$b_{ui} = \mu + b_u + b_i \quad (7-8)$$

这里 b_{ui} 表示基准算法对于评价 r_{ui} 的预测, b_u 和 b_i 分别是用户 u 和商品 i 各自对应的偏差, μ 是所有评价的平均值,即

$$\mu = \frac{1}{|K|} \sum_{(u,i) \in K} r_{ui} \quad (7-9)$$

下面介绍如何估计 b_u 和 b_i 。与机器学习中的很多算法类似,可以在推荐问题中引入损失函数,通过最小化损失函数,得到参数的最佳估计值。这里采用平方和 (sum-of-squares) 损失函数,求解如下问题:

$$\min_{b_u, b_i} \sum_{(u,i) \in K} (r_{ui} - \mu - b_u - b_i)^2 \quad (7-10)$$

与前面讨论过的算法类似,我们需要考虑模型的过拟合问题。类似地,我们也加入正则化项 (regularization),则需要求解如下问题:

$$\min_{b_u, b_i} L_b = \sum_{(u,i) \in K} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right) \quad (7-11)$$

这里 $\lambda_1 \geq 0$ 是控制正则化项权重的参数。

对于该优化问题,通常我们使用随机梯度下降算法来求解。在下面介绍基于 SVD 的模型时,将详细介绍如何使用随机梯度下降算法来求解更复杂的模型。

7.3.2 基于奇异值分解的推荐算法

在数学基础中,我们详细讨论了奇异值分解 (singular value decomposition, SVD)。这里我们讨论基于 SVD 的推荐算法。基于 SVD 的推荐算法的一个核心思想:用户和商品的特征都可以用一个低维的向量表示,而用户与商品之间的相似度可以用它们对应的向量的内积来表示。

为什么可以假设将用户和商品都用低维向量来表示呢?我们在考虑评价矩阵 R 时,发现 R 并不是杂乱无章的。这里我们以 Netflix 竞赛中的场景为例讨论。在 Netflix 竞赛中,需要考虑用户和电影之间的关系。对于每一个用户而言,他可能只对某一类电影或者某些演员出演的电影比较感兴趣。换句话说,矩阵 R 本身虽然是 $m \times n$ 维的,但是其实存在着一些内在特性使得我们可以利用。

我们从一个简单的例子开始讨论。在这个例子中,假设有用户 4 人: A、B、C 和 D; 假设有 6 部电影,记为 M1~M6。其中前面 3 部电影是喜剧片,后面 3 部电影是动作片。在这 4 个用户中, A 和 B 对喜剧片更感兴趣, C 和 D 对动作片更感兴趣。每个用户对电影进行打

分, 分数为 1~5, 越高越好。我们得到的评价矩阵 R 为:

$$R = \begin{bmatrix} 4 & 3 & 5 & 0 & 0 & 0 \\ 5 & 5 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 & 3 \end{bmatrix}$$

可以计算关于 R 的奇异值分解 $R = U\Sigma V^T$, 其中:

$$U = \begin{bmatrix} -0.65 & 0.00 & 0.75 & 0.00 \\ -0.76 & 0.00 & -0.65 & 0.00 \\ 0.00 & 0.73 & 0.00 & -0.68 \\ 0.00 & 0.68 & 0.00 & 0.73 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 10.66 & 0 & 0 & 0 \\ 0 & 10.230 & 0 & 0 \\ 0 & 0 & 1.56 & 0 \\ 0 & 0 & 0 & 1.54 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.60 & 0.00 & -0.16 & 0.00 \\ -0.54 & 0.00 & -0.64 & 0.00 \\ -0.59 & 0.00 & 0.75 & 0.00 \\ 0.00 & 0.55 & 0.00 & 0.13 \\ 0.00 & 0.62 & 0.00 & 0.60 \\ 0.00 & 0.56 & 0.00 & -0.79 \end{bmatrix}$$

我们取 U 、 V 的前两列和 Σ 中的前两个奇异值来逼近 R :

$$\hat{R} = \begin{bmatrix} -0.65 & 0.00 \\ -0.76 & 0.00 \\ 0.00 & 0.73 \\ 0.00 & 0.68 \end{bmatrix} \begin{bmatrix} 10.66 & 0 \\ 0 & 10.230 \end{bmatrix} \begin{bmatrix} -0.60 & 0.00 \\ -0.54 & 0.00 \\ -0.59 & 0.00 \\ 0.00 & 0.55 \\ 0.00 & 0.62 \\ 0.00 & 0.56 \end{bmatrix}^T = \begin{bmatrix} 4.19 & 3.76 & 4.12 & 0 & 0 & 0 \\ 4.84 & 4.34 & 4.76 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4.13 & 4.63 & 4.17 \\ 0 & 0 & 0 & 3.86 & 4.32 & 3.89 \end{bmatrix}$$

可以看出 \hat{R} 和 R 是比较接近的。但是 \hat{R} 的秩是 2。换言之, 我们只使用了矩阵 R 的 SVD 的一部分信息, 但是仍然保留了矩阵 R 中最主要的信息。

从上面的例子可以看出, 我们可以使用低秩矩阵来近似表达原始矩阵 R 。如果将每个评价 r_{ui} 都看成用户 u 对应的向量和商品 i 对应的向量的内积的话, 就可以使用低维向量来表示用户和商品。

下面使用严格的数学语言来描述。我们将低维空间的维度记为 d , 将用户 u 的低维表示记为 $\mathbf{p}_u \in \mathbb{R}^d$, 将商品 i 的低维表示记为 $\mathbf{q}_i \in \mathbb{R}^d$ 。那么我们可以用它们的内积 $\mathbf{p}_u^T \mathbf{q}_i$ 来表示用户 u 对商品 i 的喜好程度。同时, 也要考虑基准模型中的 b_u 和 b_i 。将这些因素综合考虑在一

起, 可以将预测的评价 \hat{r}_{ui} 写成:

$$\hat{r}_{ui} = \mu + b_u + b_i + \mathbf{p}_u^T \mathbf{q}_i \quad (7-12)$$

式(7-12)是基于SVD的推荐算法的基本假设。下面讨论的很多矩阵分解的推荐算法都可以视为该算法的扩展。

与基准模型类似, 我们也可通过最小化相应的损失函数来得到最优的参数 b_u 、 b_i 、 \mathbf{p}_u 、 \mathbf{q}_i 。如果我们采用平方和损失函数的话, 其对应的损失函数 L_{SVD} 为:

$$\begin{aligned} L_{\text{SVD}} &= \sum_{(u,i) \in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right) + \lambda_2 \left(\sum_u \|\mathbf{p}_u\|_2^2 + \sum_i \|\mathbf{q}_i\|_2^2 \right) \\ &= \sum_{(u,i) \in K} (r_{ui} - \mu - b_u - b_i - \mathbf{p}_u^T \mathbf{q}_i)^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right) + \lambda_2 \left(\sum_u \|\mathbf{p}_u\|_2^2 + \sum_i \|\mathbf{q}_i\|_2^2 \right) \end{aligned} \quad (7-13)$$

我们要最小化损失函数 L_{SVD} 以求得最优参数 b_u 、 b_i 、 \mathbf{p}_u 、 \mathbf{q}_i 。这里我们已经考虑了关于 \mathbf{p}_u 、 \mathbf{q}_i 的正则化项。控制参数 $\lambda_1 \geq 0$ 、 $\lambda_2 \geq 0$ 一般可以通过交叉检验得到, 以求得最优的参数 b_u 、 b_i 、 \mathbf{p}_u 、 \mathbf{q}_i 。在实际应用中, 用户根据数据的具体特征还可以为每个参数选取不同的控制参数。例如, 可以将 $\sum_u \|\mathbf{p}_u\|_2^2$ 的权重对应的控制参数设为 λ_{21} , 将 $\sum_i \|\mathbf{q}_i\|_2^2$ 权重的控制参数设为 λ_{22} , 并分别调节 λ_{21} 和 λ_{22} 的值。这样, 损失函数中 \mathbf{p}_u 和 \mathbf{q}_i 对应的正则化项就变为 $\lambda_{21} \sum_u \|\mathbf{p}_u\|_2^2 + \lambda_{22} \sum_i \|\mathbf{q}_i\|_2^2$ 。

下面我们介绍如何求解该优化问题。因为该优化问题不是机器学习中常见的凸优化(convex optimization)问题, 所以理论上不能同时求得 \mathbf{p}_u 和 \mathbf{q}_i 的最优解。一般来讲, 我们有两种算法来求解: (1) 交替最小二乘法; (2) 随机梯度下降算法。

交替最小二乘法是交替算法(alternating algorithm)的一种。交替算法的基本思想: 在一个优化问题中有多个参数需要求解时, 如果同时求解所有参数的难度太高, 那么可以先固定一部分参数(该参数集记为 P_1)的值, 而去优化剩余的参数(记为 P_2); 然后固定刚刚求得的参数 P_2 的值, 来优化 P_1 。反复使用该流程直到所有的参数都收敛到最优解。注意, 使用交替算法的时候我们一般求得的都是局部最优解(而不是全局最优解)。

具体来说, 在优化 L_{SVD} 时, 我们不能同时求解 \mathbf{p}_u 和 \mathbf{q}_i 。但是, 如果我们固定 \mathbf{p}_u , 那么求解 \mathbf{q}_i 可以直接使用最小二乘法。固定 \mathbf{q}_i , 求解 \mathbf{p}_u 也可使用同样的算法。

在随机梯度下降算法中, 逐次遍历评价矩阵 \mathbf{R} 中每个已知元素 r_{ui} , 并更新模型中的所有对应参数, 包括 b_u 、 b_i 、 \mathbf{p}_u 、 \mathbf{q}_i 。在随机梯度下降算法中, 每一步我们求解目标函数对于每个参数的梯度(或者导数), 然后沿着负梯度的方向按照一定步长更新对应的参数。

这里以参数 \mathbf{p}_u 为例来说明随机梯度算法如何工作。注意, 损失函数 L_{SVD} 考虑了集合 K 中的所有评价价值。当顺次处理评价项 r_{ui} 时, 损失函数 L_{SVD} 与 r_{ui} 相关的项 $L_{\text{SVD}}(r_{ui})$ 为:

$$L_{\text{SVD}}(r_{ui}) = (r_{ui} - \mu - b_u - b_i - \mathbf{p}_u^T \mathbf{q}_i)^2 + \lambda_1 (b_u^2 + b_i^2) + \lambda_2 (\mathbf{p}_u^T \mathbf{p}_u + \mathbf{q}_i^T \mathbf{q}_i) \quad (7-14)$$

可以计算 $L_{\text{SVD}}(r_{ui})$ 关于 \mathbf{p}_u 的梯度如下:

$$\frac{\partial L_{\text{SVD}}(r_{ui})}{\partial \mathbf{p}_u} = 2(\mathbf{r}_{ui} - \mu - b_u - b_i - \mathbf{p}_u^T \mathbf{q}_i)(-\mathbf{q}_i) + 2\lambda_2 \mathbf{p}_u \quad (7-15)$$

把 $r_{ui} - \mu - b_u - b_i - \mathbf{p}_u^T \mathbf{q}_i = r_{ui} - \hat{r}_{ui}$ 记为 e_{ui} , 则 $\frac{\partial L_{\text{SVD}}(r_{ui})}{\partial \mathbf{p}_u}$ 可记为:

$$\frac{\partial L_{\text{SVD}}(r_{ui})}{\partial \mathbf{p}_u} = -2e_{ui}\mathbf{q}_i + 2\lambda_2 \mathbf{p}_u = -2(e_{ui}\mathbf{q}_i - \lambda_2 \mathbf{p}_u) \quad (7-16)$$

在随机梯度下降算法中, 可以利用如下公式来更新参数 \mathbf{p}_u :

$$\mathbf{p}_u = \mathbf{p}_u - \gamma \frac{\partial L_{\text{SVD}}(r_{ui})}{\partial \mathbf{p}_u} = \mathbf{p}_u + 2\gamma(e_{ui}\mathbf{q}_i - \lambda_2 \mathbf{p}_u) \quad (7-17)$$

这里 $\gamma > 0$ 称为步长或者学习率 (learning rate)。

对于其他参数, 可以使用类似的技巧来更新它们。这里列出所有的更新公式而省略具体的推导过程。

$$b_u = b_u + 2\gamma(e_{ui} - \lambda_1 b_u) \quad (7-18)$$

$$b_i = b_i + 2\gamma(e_{ui} - \lambda_1 b_i) \quad (7-19)$$

$$\mathbf{q}_i = \mathbf{q}_i + 2\gamma(e_{ui}\mathbf{p}_u - \lambda_2 \mathbf{q}_i) \quad (7-20)$$

注意, 这里为所有的参数都使用了同样的步长 γ 。在实际中, 用户可以根据实际的数据对不同的参数使用不同的步长。此外, 对于不同的参数的正则化项, 还可使用不同的系数。

算法 7-1 给出了使用随机梯度下降算法训练一个完整的 SVD 推荐模型的具体步骤。

算法 7-1 适用于 SVD 模型的随机梯度下降算法

输入: 评价矩阵 \mathbf{R} 。

控制参数: 学习率 γ , 低维维度 d , 正则化参数 λ_1 、 λ_2 。

初始化用户对应的参数 $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m$ 和 $b_{u1}, b_{u2}, \dots, b_{um}$, 商品对应的向量 $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n$ 和 $b_{i1}, b_{i2}, \dots, b_{in}$ 。

计算评价的平均值 $\mu = \frac{1}{|K|} \sum_{(u,i) \in K} r_{ui}$ 。

while 停止标准没有满足

对于所有的 $(u, i) \in K$

计算 $\hat{r}_{ui} = \mu + b_u + b_i + \mathbf{p}_u^T \mathbf{q}_i$

计算错误 $e_{ui} = \hat{r}_{ui} - r_{ui}$

更新相关参数:

$$b_u = b_u + 2\gamma(e_{ui} - \lambda_1 b_u)$$

$$b_i = b_i + 2\gamma(e_{ui} - \lambda_1 b_i)$$

$$p_u = p_u + 2\gamma(e_{ui} q_i - \lambda_2 p_u)$$

$$q_i = q_i + 2\gamma(e_{ui} p_u - \lambda_2 q_i)$$

在算法 7-1 中, 可以使用不同的停止标准。例如, 在实际中, 可以单独划分一个检验集(validation set)。当所学习到的模型在检验集上的性能达到要求(如在检验集上的 RMSE 小于规定值)时, 可以认为停止标准达到了。

讨论

在第 3 章中我们讨论了奇异值分解(SVD)。在本节所讨论的推荐算法中, 虽然这里的算法也称为基于 SVD 的模型, 但是它们之间有一些不同:

- 在标准的 SVD 中, 整个矩阵都是已知的; 而在推荐问题中, 通常只知道矩阵的一部分, 并且经常是稀疏的, 这样矩阵计算中已有的求解 SVD 的算法不能直接适用于推荐问题。
- 在推荐问题中, 由于评价矩阵 R 太稀疏, 因此必须非常注意过拟合问题。在前面的讨论中, 引入了正则化项, 在实际使用中也需要读者根据数据的具体情况选择合适的参数值。
- 推荐算法中的 SVD 模型更加接近于用一个低秩的矩阵来逼近矩阵 R :

$$R \approx P^T Q = \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_m^T \end{bmatrix} [q_1, q_2, \dots, q_n] \quad (7-21)$$

这里

$$P = [p_1, p_2, \dots, p_m] \in \mathbb{R}^{d \times m}, \quad Q = [q_1, q_2, \dots, q_n] \in \mathbb{R}^{d \times n} \quad (7-22)$$

根据式 (7-21), 我们把 R 分解为两个低秩矩阵的积, 所以有些文献(如参考文献[28])中也把这种矩阵分解称为 UV 分解。为了避免过拟合, 通常还进一步对矩阵 P 和 Q 做出了一些限制。在上面讨论的算法中, 为 p_u 和 q_i 都引入了正则化项。

7.3.3 基于 SVD 推荐算法的变体

前面讨论的基于奇异值分解的基本算法原理简单, 也易于实现。但是在实际中, 如果数据较为复杂, 并且数据和该算法的基本假设存在较大差异的话, 则很难取得很好的效果。在本节我们进一步介绍基于 SVD 模型的变体, 包括 AFM 模型、翻转的 AFM 模型、ASVD 模型、翻转的 ASVD 模型, 以及引入时间信息的矩阵分解模型。

1. AFM 模型

AFM 模型一般称为不对称因子模型 (asymmetric factor model, AFM)。在该模型中, 只有与商品有关的参数。具体而言, 对于每个评价 r_{ui} , 我们假设可以用下面的公式来表示:

$$\hat{r}_{ui} = \mu + b_u + b_i + \left(\frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} q_j^{(0)} \right)^T q_i \quad (7-23)$$

其关键性的假设: 对于每件商品, 还是使用 $q_i \in \mathbb{R}^d$ 来给出它对应的低维表示; 对于每个用户, 我们假设对应的低维向量可以由其在训练集中评价过的商品来表示。这里对于用户 u , 将其在训练集中评价过的商品的集合记为 $I(u)$, 并用 $I(u)$ 中的商品来表示用户 u 。用户 u 的低维表示为:

$$p_u = \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} q_j^{(0)} \quad (7-24)$$

对于相关的商品 $i \in I(u)$, 用向量 $q_i^{(0)}$ 表示该商品对于用户 u 对应向量的贡献。此外, 还是用 $\frac{1}{\sqrt{|I(u)|}}$ 来对用户 u 进行标准化处理, 这里 $|I(u)|$ 是用户 u 评价过的商品的总数。注意,

这里对于每件商品 i , 构建了两个不同的向量。

- $q_i \in \mathbb{R}^d$: 用来表示商品 i 在低维空间的表示;
- $q_i^{(0)} \in \mathbb{R}^d$: 用来表示商品 i 对于相关用户对应的低维向量的贡献。

在实际中, 我们要从训练数据中为每件商品学出 q_i 和 $q_i^{(0)}$ 。

2. 翻转的 AFM 模型

在 AFM 模型中, 我们将用户在低维空间的表示也用相关商品的低维表示来表示。类似地, 也可以将 AFM “翻转” 过来: 我们只有用户的低维表示, 商品的低维表示由对应用户的低维表示决定。

具体来说, 对于每个用户 u , 构建两个向量。

- $p_u \in \mathbb{R}^d$: 用来表示用户 u 的低维表示;
- $p_u^{(0)} \in \mathbb{R}^d$: 用来表示用户 u 对于相关商品对应的低维向量的贡献。

这样, 对于商品 i , 可以使用 $p_u^{(0)}$ 来表示:

$$q_i = \frac{1}{\sqrt{|U(i)|}} \sum_{v \in U(i)} p_v^{(0)} \quad (7-25)$$

这里 $U(i)$ 是训练集中评价过商品 i 的用户集合, $|U(i)|$ 是评价过商品 i 的用户总数。

这样每个评价 r_{ui} 的预测值可以用如下的公式来表示:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T \left(\frac{1}{\sqrt{|U(i)|}} \sum_{v \in U(i)} p_v^{(0)} \right) \quad (7-26)$$

AFM 模型和翻转的 AFM 模型都可以使用随机梯度下降算法来求解。下面我们介绍更加通用的 ASVD 模型, 并介绍对应的随机梯度下降算法。由于 AFM 模型和翻转的 AFM 模型是 ASVD 模型的特例, 因此可以将所介绍的随机梯度下降算法简化, 这样就可以直接求解了。

3. ASVD 模型

在 ASVD 模型中, 综合考虑了 SVD 模型和 AFM 模型。它也称为 SVD++ 模型。一般而言, 在很多场合下, ASVD 模型 (或者 SVD++ 模型) 的性能要优于前面讨论的基本 SVD 模型。

具体而言, 在 ASVD 模型中, 用户 u 的低维表示为:

$$\mathbf{p}_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} \mathbf{q}_j^{(0)} \quad (7-27)$$

与前面的 SVD 模型相比, 对于用户 u , 添加了来自于他/她所评价的商品的信息。这样, 我们希望能够更加准确地描述用户与商品之间的关系。

这样, 可以将 r_{ui} 的预测值 \hat{r}_{ui} 写成:

$$\hat{r}_{ui} = \mu + b_u + b_i + \left(\mathbf{p}_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} \mathbf{q}_j^{(0)} \right)^T \mathbf{q}_i \quad (7-28)$$

下面介绍适用于 ASVD 模型的随机梯度下降算法。与 SVD 模型类似, 我们假设使用平方和损失函数, 并考虑了各参数的正则化项:

$$\begin{aligned} L_{\text{ASVD}} &= \sum_{(u,i) \in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right) + \lambda_2 \left(\sum_u \|\mathbf{p}_u\|_2^2 + \sum_i \|\mathbf{q}_i\|_2^2 \right) + \lambda_3 \sum_i \|\mathbf{q}_j^{(0)}\|_2^2 \\ &= \sum_{(u,i) \in K} \left(r_{ui} - \mu - b_u - b_i - \left(\mathbf{p}_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} \mathbf{q}_j^{(0)} \right)^T \mathbf{q}_i \right)^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right) \\ &\quad + \lambda_2 \left(\sum_u \|\mathbf{p}_u\|_2^2 + \sum_i \|\mathbf{q}_i\|_2^2 \right) + \lambda_3 \sum_i \|\mathbf{q}_j^{(0)}\|_2^2 \end{aligned} \quad (7-29)$$

这里 $\lambda_1 \geq 0$ 、 $\lambda_2 \geq 0$ 、 $\lambda_3 \geq 0$, 它们是控制正则化项权重的参数。

在随机梯度下降算法中, 我们顺次遍历评价矩阵 \mathbf{R} 中每个已知元素 r_{ui} , 并更新模型中的所有对应参数, 包括 b_u 、 b_i 、 \mathbf{p}_u 、 \mathbf{q}_i 、 $\mathbf{q}_i^{(0)}$ 。当我们顺次处理评价项 r_{ui} 时, 其对应的损失函数 $L_{\text{ASVD}}(r_{ui})$ 为:

$$\begin{aligned} L_{\text{ASVD}}(r_{ui}) &= \left(r_{ui} - \mu - b_u - b_i - \left(\mathbf{p}_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} \mathbf{q}_j^{(0)} \right)^T \mathbf{q}_i \right)^2 + \lambda_1 (b_u^2 + b_i^2) \\ &\quad + \lambda_2 \left(\mathbf{p}_u^T \mathbf{p}_u + \mathbf{q}_i^T \mathbf{q}_i \right) + \lambda_3 \sum_{i \in I(u)} \|\mathbf{q}_j^{(0)}\|_2^2 \end{aligned} \quad (7-30)$$

这里逐个计算 $L_{\text{ASVD}}(r_{ui})$ 对于各个参数的偏导数。如前面的讨论, 记 $e_{ui} = r_{ui} - \mu - b_u - b_i -$

$\left(p_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} q_j^{(0)} \right)^T q_i$, 则有:

$$\frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial b_u} = 2e_{ui}(-1) + 2\lambda_1 b_u \quad (7-31)$$

$$\frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial b_i} = 2e_{ui}(-1) + 2\lambda_1 b_i \quad (7-32)$$

$$\frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial p_u} = 2e_{ui}(-1)q_i + 2\lambda_2 p_u \quad (7-33)$$

$$\frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial q_i} = 2e_{ui}(-1) \left(p_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} q_j^{(0)} \right) + 2\lambda_2 q_i \quad (7-34)$$

$$\forall j \in I(u), \frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial q_j^{(0)}} = 2e_{ui}(-1) \frac{1}{\sqrt{|I(u)|}} q_i + 2\lambda_3 q_j^{(0)} \quad (7-35)$$

在随机梯度下降算法中, 我们都是用负导数来更新对应的参数。因此, 在 ASVD 模型中, 使用如下的公式来更新诸参数:

$$b_u = b_u - \gamma \frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial b_u} = b_u + 2\gamma(e_{ui} - \lambda_1 b_u) \quad (7-36)$$

$$b_i = b_i - \gamma \frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial b_i} = b_i + 2\gamma(e_{ui} - \lambda_1 b_i) \quad (7-37)$$

$$p_u = p_u - \gamma \frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial p_u} = p_u + 2\gamma(e_{ui}q_i - \lambda_2 p_u) \quad (7-38)$$

$$q_i = q_i - \gamma \frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial q_i} = q_i + 2\gamma \left(e_{ui} \left(p_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} q_j^{(0)} \right) - \lambda_2 q_i \right) \quad (7-39)$$

$$\forall j \in I(u), q_j^{(0)} = q_j^{(0)} - \gamma \frac{\partial L_{\text{ASVD}}(r_{ui})}{\partial q_j^{(0)}} = q_j^{(0)} + 2\gamma \left(\frac{e_{ui}}{\sqrt{|I(u)|}} q_i - \lambda_3 q_j^{(0)} \right) \quad (7-40)$$

ASVD 模型还能进一步扩展。在很多实际应用中, 有所谓的隐式反馈 (implicit feedback)。如果我们在淘宝上购物, 除了用户购买的商品外, 用户点击的商品也反映了用户的兴趣。在很多实际应用中, 考虑隐式反馈的信息很容易获得, 而直接的信息 (如购买) 却很少。下面我们讨论一种考虑隐式反馈的 ASVD 模型。记 $I_y(u)$ 为用户 u 所提供的隐式反馈所对应的商品集, 在 ASVD 模型的基础上, 可以将用户 u 在低维空间的表示写成如下形式:

$$\mathbf{p}_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} \mathbf{q}_j^{(0)} + \frac{1}{\sqrt{|I_{if}(u)|}} \sum_{j \in I_{if}(u)} \mathbf{q}_j^{(1)} \quad (7-41)$$

在式(7-41)中, 第二项 $\frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} \mathbf{q}_j^{(0)}$ 对应于直接反馈的商品的贡献, 第三项 $\frac{1}{\sqrt{|I_{if}(u)|}} \sum_{j \in I_{if}(u)} \mathbf{q}_j^{(1)}$ 对应于隐式反馈的商品的贡献。而对商品 i 而言, 需要考虑以下3项。

- $\mathbf{q}_i \in \mathbb{R}^d$: 用来表示商品 i 在低维空间的表示。
- $\mathbf{q}_i^{(0)} \in \mathbb{R}^d$: 用来表示商品 i 对于直接相关用户对应的低维向量的贡献。
- $\mathbf{q}_i^{(1)} \in \mathbb{R}^d$: 用来表示商品 i 对于隐式反馈相关用户对应的低维向量的贡献。

这样, 可以将 r_{ui} 的预测值 \hat{r}_{ui} 写成:

$$\hat{r}_{ui} = \mu + b_u + b_i + \left(\mathbf{p}_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} \mathbf{q}_j^{(0)} + \frac{1}{\sqrt{|I_{if}(u)|}} \sum_{j \in I_{if}(u)} \mathbf{q}_j^{(1)} \right)^T \mathbf{q}_i \quad (7-42)$$

类似地, 利用随机梯度下降算法, 也可以求解该模型。

讨论 与简单的 SVD 模型相比, ASVD 模型引入的参数更多。为了避免过拟合, 我们更倾向于选择较小的 d 值, 即低维空间的维度较低, 从而控制模型的复杂度。

4. 翻转的 ASVD 模型

与前面讨论类似, 我们也有翻转的 ASVD 模型。在翻转的 ASVD 模型中, 对于每个用户, 使用 \mathbf{p}_u 来表示他/她的低维表示, 而商品 i 的低维表示为:

$$\mathbf{q}_i + \frac{1}{\sqrt{|U(i)|}} \sum_{v \in U(i)} \mathbf{p}_v^{(0)} \quad (7-43)$$

这里为每个用户 v 定义了 $\mathbf{p}_v^{(0)}$, 表示他/她对相关商品的贡献。这样, 可以将 r_{ui} 的预测值 \hat{r}_{ui} 写成:

$$\hat{r}_{ui} = \mu + b_u + b_i + \mathbf{p}_u^T \left(\mathbf{q}_i + \frac{1}{\sqrt{|U(i)|}} \sum_{v \in U(i)} \mathbf{p}_v^{(0)} \right) \quad (7-44)$$

此外, 也可以引入隐式反馈, 将 r_{ui} 的预测值 \hat{r}_{ui} 写成:

$$\hat{r}_{ui} = \mu + b_u + b_i + \mathbf{p}_u^T \left(\mathbf{q}_i + \frac{1}{\sqrt{|U(i)|}} \sum_{v \in U(i)} \mathbf{p}_v^{(0)} + \frac{1}{\sqrt{|U_{if}(i)|}} \sum_{v \in U_{if}(i)} \mathbf{p}_v^{(1)} \right) \quad (7-45)$$

这里将在隐式反馈中将商品 i 相关的用户集合记为 $U_{if}(i)$ 。

类似地, 对于翻转的 ASVD 模型, 也可以导出相应的导数, 并利用随机梯度下降算法来

求解。其基本过程和公式与上一节中的讨论非常类似，这里就不再重复了。

5. 引入时间信息的模型

在前面的讨论中，我们介绍了如何利用直接反馈和隐式反馈。这里我们介绍如何进一步引入时间信息。在很多应用中，如果评价数据与时间信息存在某些周期性的关系，那么引入时间特征可以显著地提高推荐算法的性能。

在本节中，我们从简单的例子（无矩阵分解的基准算法）开始，讨论如何引入时间信息，然后介绍如何将时间信息引入到矩阵分解的算法中。在这里我们强调，时间信息的引入依赖于问题的具体定义和实际数据，读者需要灵活应用时间信息以得到最优性能。

首先我们讨论如何在无矩阵分解的基准算法中加入时间信息。在实际中，推荐系统的实际表现和时间关系很大。例如，在前面的讨论中，每件商品都有对应的 b_i 。在电影推荐中，很多电影对应的 b_i 随着电影的档期发生了明显的变化。在购物推荐中，用户的喜好在不同的季节有着明显的变化。因此，在很多实际应用中， b_i 和 b_u 都是关于时间的函数。那么，在无矩阵分解的基准方法中，如果我们引入时间信息，则可将 b_{ui} 表示为

$$b_{ui} = \mu + b_u(t_{ui}) + b_i(t_{ui}) \quad (7-46)$$

这里 t_{ui} 是评价 r_{ui} 产生的时间。

如果我们将 b_i 和 b_u 都看成是关于时间 t_{ui} 的连续函数，那么需要足够的数据才能准确地估计对应的函数。此外，直接将 b_i 和 b_u 都当做是关于时间 t_{ui} 的连续函数会导致推荐模型的复杂度较高，在实际中容易过拟合。

在实际中，一种有效的方法是将时间轴划分为若干段，并将 b_i 和 b_u 在每段的值视为一恒定值。举一个简单的例子，在网上购物时，有的用户习惯于上午或下午购物，而有的用户更偏向于下班之后的晚上购物。这样，我们可以把时间轴分为3段：上午（记为时间段1）、下午（记为时间段2）及晚上（记为时间段3），从而分别考虑 b_i 和 b_u 在不同时间段的值。这样就可将 b_{ui} 表示为：

$$b_{ui} = \mu + b_u(t_{ui}) + b_i(t_{ui}) = \mu + b_u + b_{ui} + b_i + b_{ii} \quad (7-47)$$

这里

$$b_u(t_{ui}) = b_u + b_{ui}, \quad b_{ui} \in \{b_{u1}, b_{u2}, b_{u3}\} \quad (7-48)$$

$$b_i(t_{ui}) = b_i + b_{ii}, \quad b_{ii} \in \{b_{i1}, b_{i2}, b_{i3}\} \quad (7-49)$$

这里 b_{u1} 、 b_{u2} 、 b_{u3} 分别代表上午、下午和晚上对 b_u 的调节值。根据当时的时间 t_{ui} ，我们从 b_{u1} 、 b_{u2} 、 b_{u3} 中选出对应的项来调节 b_u 的值。类似地， b_{i1} 、 b_{i2} 、 b_{i3} 分别代表上午、下午和晚上对 b_i 的调节值。

这种做法的好处：（1）需要估计的参数值一般不多，模型的复杂度也不高；（2）能够得到足够多的数据来估计参数值；（3）在实际中，一般能够取得比较理想的性能。但缺点是需要根据数据和问题手动地确定分段。

接下来我们考虑如何在矩阵分解模型中进一步加入时间信息。与前面的基准方法类似，

对于商品 i ，除了其低维表示 q_i 外，还要考虑其随着时间变化而变化的项。简而言之，可以将商品 i 的新的低维表示记为：

$$q_i + q_i^{(t)}$$

其中 $q_i^{(t)}$ 是关于时间的函数。

对于用户 u ，也可以使用类似的项来调节用户的低维表示：

$$p_u + p_u^{(t)}$$

其中 $p_u^{(t)}$ 是关于时间的函数。

在实际应用中，与前面讨论的基准方法类似，我们可以：（1）将 $q_i^{(t)}$ 和 $p_u^{(t)}$ 考虑成关于时间的连续函数；（2）将时间轴分为若干段，但是每段都对应一个恒定值。这里举一个简单的例子来说明如何使用第二种方法。在购物网站中，一般而言，用户在周末和非周末的购买行为和习惯会有所不同，因此我们将时间分为周末和非周末，并分别用 p_{u1} 和 p_{u2} 来表示周末和非周末的调节量。于是，用户 u 的低维表示为：

$$p_u + p_{ui}, \quad p_{ui} \in \{p_{u1}, p_{u2}\}$$

与前面的讨论类似，可以引入对应的损失函数（包括正则化项）并求解。

7.4 基于邻域的推荐算法

在基于矩阵分解的推荐算法兴起之前，最常用的推荐算法是基于邻域的推荐算法（neighborhood-based recommendation）。基于邻域的推荐算法的基本思想：当我们要向用户 u 推荐商品 i 时，找出与用户 u 相似的用户，或者与商品 i 相似的商品，从相似的用户或者商品的历史数据中推断用户 u 对商品 i 的喜好程度。基于邻域的推荐算法也称为基于记忆的推荐算法（memory-based recommendation）或者基于启发式的推荐算法（heuristic-based recommendation）。

在基于邻域的推荐算法中，主要利用用户-用户之间的相似度或者商品-商品之间的相似度来预测未知的评价值 r_{ui} 。具体而言，可以从寻找相似的用户出发，也可以从寻找相似的商品开始处理。因此，算法分为如下两类。

- 基于用户的邻域推荐算法：如果用户 u 和用户 v 对很多商品的喜好一致，那么可以使用用户 u 对于某一商品 i 的已知评价来推断用户 v 对该商品的评价。
- 基于商品的邻域推荐算法：如果商品 i 和商品 j 在很多用户的评价中都比较一致，那么可以使用用户 u 对商品 i 的已知评价来推断用户 u 对商品 j 的评价。

与基于矩阵分解的推荐算法相比，基于邻域的推荐算法在原理上十分简单直观，并且易于实现。基于邻域的推荐算法的主要控制参数包括邻域的大小和相似度度量的选择。因此，需要学习的参数较少，模型的复杂度也比较低。此外，当有了新的数据需要更新模型时，只需要更新与新数据相关的用户和商品即可，不需要重新训练整个模型。在实际中，基于邻域

的推荐算法更容易解释和理解。而基于矩阵分解的推荐算法依赖于用户和商品的低维表示，模型不容易解释。下面我们具体讨论基于用户的邻域推荐算法和基于商品的邻域推荐算法。

7.4.1 基于用户的邻域推荐算法

在推荐系统中，我们面临的首要问题是：对于用户 u 和商品 i ，如何估计用户 u 对商品 i 的喜好程度 r_{ui} ？在基于用户的邻域推荐算法（user-based neighborhood recommendation）中，我们的基本思想是从已经评价过商品 i 的用户集 $U(i)$ 中，找出与用户 u 相似的用户，并利用这些用户对商品 i 的评价来推断用户 u 对商品 i 的喜好程度。简而言之，就是从商品 i 相关的用户群中找出与用户 u 相似的用户，从而进一步推断 r_{ui} 。

严格地讲，就是在评价过商品 i 的用户集 $U(i)$ 中，寻找与用户 u 相似的 k 个用户组成的集合（记为最近邻集 $N_i(u)$ ），并利用 $N_i(u)$ 来估计 r_{ui} ：

$$\hat{r}_{ui} = \frac{1}{|N_i(u)|} \sum_{v \in N_i(u)} r_{vi} \quad (7-50)$$

式 (7-50) 简洁明了，在实际中是较实用的公式。其缺陷是没有仔细考虑 $N_i(u)$ 中的每个用户和用户 u 之间的相似程度。如果考虑每个用户 $v \in N_i(u)$ 与用户 u 的相似度（记为 w_{uv} ），则可以使用如下公式来估计 r_{ui} ：

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i(u)} w_{uv} r_{vi}}{\sum_{v \in N_i(u)} |w_{uv}|} \quad (7-51)$$

这里在分母中使用了 $|w_{uv}|$ ，因为在很多情况下 w_{uv} 可能为负值。为了保证 \hat{r}_{ui} 落在正常的范围之内，我们使用了 $|w_{uv}|$ 。

评价的标准化

上面的式 (7-50) 和式 (7-51) 直接根据邻域 $N_i(u)$ 中的用户评价来计算加权平均值。在前面的讨论中，我们知道每个用户对应不同的 b_u 项：有的用户更加愿意给出较高的评价，有些用户则倾向于给出较低的评价。因此，我们希望在综合多个用户的评价时，能够消除用户的个人偏见。一个常用的办法是将所有的评价先标准化（normalization），然后综合。这里我们主要介绍两种常用的标准化方法：基于平均值的标准化和 Z 分值标准化。

在基于平均值的标准化中，我们将评分 r_{ui} 与对应的平均值相比较，将所得的差值作为新的评分。这里的平均值可以是用户 u 评分的平均值，也可以是商品 i 所有的评分的平均值。下面我们具体讨论在基于用户的邻域方法中如何使用基于平均值的标准化。

在基于用户的邻域方法中，我们将 r_{ui} 与用户 u 的评分的平均值 \bar{r}_u 比较，得到标准化之后的评分：

$$h(r_{ui}) = r_{ui} - \bar{r}_u \quad (7-52)$$

这里 $h(r_{ui})$ 是标准化之后的评分， \bar{r}_u 为用户 u 当前评分的平均值：

$$\bar{r}_u = \frac{1}{|I(u)|} \sum_{i \in I(u)} r_{ui} \quad (7-53)$$

使用前面的公式计算综合多个用户的评价之后,还需要将所得的评价分数转化到原始的评价空间中。这里我们需要重新加上 \bar{r}_u 。因此, r_{ui} 的预测值可以表示为:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N_i(u)} w_{uv} (r_{vi} - \bar{r}_v)}{\sum_{v \in N_i(u)} |w_{uv}|} \quad (7-54)$$

可以看出,这里只是简单地使用每个用户的评价的平均值来对用户的原始评价进行调整。

在很多实际应用中,我们除了考虑用户评分的平均值外,还要考虑用户评分时的偏差。假设我们有两个用户: A 和 B。用户 A 和用户 B 的评分均值都是 3 分,但是用户 A 基本上对所有的商品的评价都是 3 分(分数为 1 分~5 分,越高越好),而用户 B 的评分却从 1 分到 5 分都有。如果对于同一件商品,用户 A 和用户 B 都给出了 5 分,则可以认为用户 A 对该商品的满意程度高于用户 B。因此,在对评分进行标准化处理时,我们除了要考虑评分的均值外,还需要考虑评分的“散布程度”,就是评分的方差。注意,在基于用户的邻域算法和基于商品的邻域算法中,Z 分值标准化的具体实施稍有不同。

在基于用户的邻域算法中,使用如下公式来修正 r_{ui} :

$$h(r_{ui}) = \frac{r_{ui} - \bar{r}_u}{\sigma_u} \quad (7-55)$$

这里 σ_u 是用户 u 评分的标准差。采用 Z 分值标准化之后,预测 r_{ui} 的公式则为:

$$\hat{r}_{ui} = \bar{r}_u + \sigma_u \frac{\sum_{v \in N_i(u)} w_{uv} \frac{(r_{vi} - \bar{r}_v)}{\sigma_v}}{\sum_{v \in N_i(u)} |w_{uv}|} \quad (7-56)$$

如果一个用户的评价太少,则会导致其对应的方差 $\sigma_u = 0$,导致无法使用 Z 分值标准化。一般来讲,如果用户或者商品的评价足够多的话,那么使用 Z 分值标准化能够得到更好的结果。特别是如果评分 r_{ui} 的取值范围较大的话,使用方差进行处理是非常合理的。但是使用 Z 分值标准化时有可能导致预测值 \hat{r}_{ui} 的取值超过原始定义的空间。例如,原来 $r_{ui} \in \{1, 2, 3, 4, 5\}$,使用 Z 分值之后 \hat{r}_{ui} 有可能是 6。

在实际应用中,以上讨论的这两种方法在很多时候性能都比较相近,因此要根据数据的具体情况采用相应的标准化方法。

我们在第 4 章中讨论了数据的标准化处理。这里的讨论与前面的讨论在原理上是一致的,但是这里的讨论更适用于推荐问题。

下面我们通过一个简单的例子来说明如何使用基于用户的邻域推荐算法来进行推荐。

例 7-1 在这个简化的例子中, 我们有 4 个用户, 即 $U1 \sim U4$; 有 4 部电影, 即 $M1 \sim M4$ 。用户对电影的评分为 1~5, 见表 7-1。这里我们要根据已有的评分数据, 估计用户 $U2$ 对电影 $M2$ 的评分。

表 7-1 使用基于用户的邻域推荐算法来估计电影评分

	M1	M2	M3	M4
U1	5	3	1	2
U2	1	?	4	3
U3	2	5	3	
U4	5	4	5	2

对于用户 $U2$, 假设与其最相似的两个用户是 $U3$ 和 $U4$, 并假设其相似度为 $w_{U2U3} = 0.8$ 、 $w_{U2U4} = 0.5$ 。当 $k = 2$ 时, 我们利用式 (7-51) 估计 $U2$ 对电影 $M2$ 的评分为:

$$\hat{r}_{U2M2} = \frac{0.8 \times 5 + 0.5 \times 4}{0.8 + 0.5} \approx 4.62$$

在算法实践中, 其中一个关键参数是邻域的大小, 即参数 k 的值。当 k 值太小时, 我们只使用了非常有限的信息, 导致预测的精度会较低; 当 k 值增大时, 我们考虑了更多的用户信息, 精度会进一步提高。但是当 k 值较大时, 用户 u 的邻域信息在某种程度上被“稀释”了, 或者换句话说, 有更多的噪声或者无关用户“混入”了邻域, 使得推荐的精度下降。由于具体 k 值的选取高度依赖于具体的数据, 因此在实际中我们一般采用交叉检验来确定最优的 k 值。

在算法的具体实现中, 我们需要为每个用户保存与其最相似的 k 个用户组成的集合。在实际中, 我们通常采用过滤的方法来保存最相似的用户-用户或者商品-商品相似度。主要有如下两种策略。

(1) 保存最相似的 k_0 个用户 (或者商品) 及对应的相似度。这里的 $k_0 \geq k$ 。我们一般要选择较大的 k_0 值, 这样在稍后选择 k 值时会有更多的余地。

(2) 如果用户与用户之间的相似度大于一个事先选定的阈值, 则保持; 否则直接丢弃。

7.4.2 基于商品的邻域推荐算法

在基于商品的邻域推荐算法中, 我们实际上翻转了用户和商品之间的关系。在估计用户 u 对商品 i 的喜好程度 r_{ui} 时, 其基本思想是首先考虑用户 u 评价过的所有商品, 从这些商品中选出与商品 i 最相似的 k 件商品, 并将该商品集记为 $N_u(i)$ 。然后, 考虑用户 u 对 $N_u(i)$ 中每件商品 j 的评价 r_{uj} , 将对 r_{ui} 的估计写为 r_{uj} 的线性组合:

$$\hat{r}_{ui} = \frac{1}{|N_u(i)|} \sum_{j \in N_u(i)} r_{uj} \quad (7-57)$$

类似地,也可以考虑导入权重 w_{ij} (商品 i 和商品 j 之间的相似度记为 w_{ij}), 则 r_{ui} 的估计可以表示为:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u(i)} w_{ij} r_{uj}}{\sum_{j \in N_u(i)} |w_{ij}|} \quad (7-58)$$

与基于用户的邻域推荐算法类似,也同样可以对商品的评价引入基于平均值的标准化和 Z 分值标准化,并对公式中的 r_{uj} 进行调整。

将 \bar{r}_i 记为已知的关于商品 i 的评价的平均值:

$$\bar{r}_i = \frac{1}{|U(i)|} \sum_{u \in U(i)} r_{ui} \quad (7-59)$$

使用基于商品平均值的标准化之后,评分 r_{ui} 可以表示为:

$$h(r_{ui}) = r_{ui} - \bar{r}_i \quad (7-60)$$

因此,在基于商品的邻域推荐算法中, r_{ui} 的预测值可以表示为:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in N_u(i)} w_{ij} (r_{uj} - \bar{r}_j)}{\sum_{j \in N_u(i)} |w_{ij}|} \quad (7-61)$$

在基于商品的邻域推荐算法中,使用 Z 分值对 r_{ui} 进行标准化之后为:

$$h(r_{ui}) = \frac{r_{ui} - \bar{r}_i}{\sigma_i} \quad (7-62)$$

这里 σ_i 是商品 i 所收到的评分的标准差。在采用 Z 分值之后,预测 r_{ui} 的公式为:

$$\hat{r}_{ui} = \bar{r}_i + \sigma_i \frac{\sum_{j \in N_u(i)} w_{ij} \frac{(r_{uj} - \bar{r}_j)}{\sigma_j}}{\sum_{j \in N_u(i)} |w_{ij}|} \quad (7-63)$$

例 7-2 类似地,我们也可以用基于商品的邻域推荐算法来估计表 7-1 中用户 U2 对电影 M2 的评分。对于电影 M2,假设与其最相似的两部电影 M1 和 M3,并假设其相似度为 $w_{M1M2} = 0.4$ 、 $w_{M2M3} = 0.7$ 。当 $k=2$ 时,利用式 (7-58) 估计用户 U2 对电影 M2 的评分为:

$$\hat{r}_{U2M2} = \frac{0.4 \times 1 + 0.7 \times 4}{0.4 + 0.7} \approx 2.91$$

7.4.3 混合算法

我们也可以将基于矩阵分解的模型和基于邻域的推荐算法结合起来。一种比较简单的方法:先使用矩阵分解算法得到用户和商品的低维表示 p_u 和 q_i , 然后利用 p_u 和 q_i 计算相似度和邻域。

7.4.4 相似度的计算

在构建基于邻域的推荐算法时，一个核心问题是如何计算相似度（similarity）。这里的相似度既包括用户与用户之间的相似度，也包括商品与商品之间的相似度。相似度的计算不但决定了邻域如何确定，也决定了邻域中每个样本的权重。因此，相似度的选择是基于邻域的推荐算法的核心部分。在本节，我们介绍几种常用的相似度度量，包括余弦相似度、Jaccard 相似度、Pearson 相关系数和 Spearman 秩相关系数。在下面的讨论中，我们以用户-用户相似度为例进行讨论。商品-商品相似度可以通过将适用于用户-用户相似度的公式进行简单的修改得到。

1. 余弦相似度

余弦相似度是常用的关于两个向量之间相似度的度量。其具体的定义如下：

$$\cos(\mathbf{x}_a, \mathbf{x}_b) = \frac{\mathbf{x}_a^T \mathbf{x}_b}{\|\mathbf{x}_a\|_2 \|\mathbf{x}_b\|_2} \quad (7-64)$$

这里向量 $\mathbf{x}_a, \mathbf{x}_b \in \mathbb{R}^d$ 。实质上 $\cos(\mathbf{x}_a, \mathbf{x}_b)$ 是向量 $\mathbf{x}_a, \mathbf{x}_b$ 在向量空间的夹角的余弦值。

在推荐问题中，可以利用以往的历史评价记录来为用户和商品构造向量。首先考虑如何计算用户的相似度。计算商品的相似度可以使用类似的方法得到。对于用户 u ，可以构建一个维度为 n （商品总数目）的向量 \mathbf{x}_u 如下：

$$\mathbf{x}_u = [x_{u1}, x_{u2}, \dots, x_{un}]^T, x_{ui} = \begin{cases} r_{ui}, & \text{如果用户 } u \text{ 评价过商品 } i \\ 0, & \text{否则} \end{cases} \quad (7-65)$$

这样，我们就可以使用余弦相似度来计算用户 u 和用户 v 之间的相似度 $CS(u, v)$ ：

$$CS(u, v) = \cos(\mathbf{x}_u, \mathbf{x}_v) = \frac{\sum_{i \in I(u) \cap I(v)} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I(u)} r_{ui}^2 \sum_{i \in I(v)} r_{vi}^2}} \quad (7-66)$$

在上面的公式中，我们在计算余弦相似度时考虑了所有的商品。另一种计算余弦相似度的方法是在分母中只考虑用户 u 和用户 v 共同评价过的商品，对应的余弦相似度的定义为：

$$CS(u, v) = \frac{\sum_{i \in I(u) \cap I(v)} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I(u) \cap I(v)} r_{ui}^2 \sum_{i \in I(u) \cap I(v)} r_{vi}^2}} \quad (7-67)$$

2. Jaccard 相似度

假设我们有两个集合 S 和 T ，则它们的 Jaccard 相似度定义为：

$$J(S, T) = \frac{|S \cap T|}{|S \cup T|} \quad (7-68)$$

换言之，就是它们的交集的大小和它们并集大小的比例。

在使用 Jaccard 相似度来计算用户 u 和用户 v 之间的相似度时, 可以直接计算用户 u 评价过的商品集 $I(u)$ 和用户 v 评价过的商品集 $I(v)$ 之间的 Jaccard 相似度作为两者之间的相似度:

$$J(u, v) = \frac{|I(u) \cap I(v)|}{|I(u) \cup I(v)|} \quad (7-69)$$

类似地, 商品 i 和商品 j 之间的 Jaccard 相似度为:

$$J(i, j) = \frac{|U(i) \cap U(j)|}{|U(i) \cup U(j)|} \quad (7-70)$$

使用 Jaccard 相似度时, 因为我们没有考虑不同的评分, 所以损失了一些信息。但是, 如果在推荐系统中 r_{ui} 是 0 或者 1 的形式, 如我们只有用户购买商品与否的信息, 那么 Jaccard 相似度是一个较好的选择。

3. Pearson 相关系数

Pearson 相关系数 (Pearson correlation coefficient) 是一种在推荐算法中广泛使用的相似度度量。在计算 $CS(u, v)$ 时, 我们没有考虑每个用户的评价的平均值和方差的影响。因此, 我们可以使用 Pearson 相关系数来度量两个用户之间的相似度:

$$PC(u, v) = \frac{\sum_{i \in I(u) \cap I(v)} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I(u) \cap I(v)} (r_{ui} - \bar{r}_u)^2 \sum_{i \in I(u) \cap I(v)} (r_{vi} - \bar{r}_v)^2}} \quad (7-71)$$

当我们考虑用户 u 和用户 v 之间的 Pearson 相关系数时, 只考虑 $I(u) \cap I(v)$ 中的商品, 即他们都评价过的商品, 而不是所有商品。因此, 在分母中, 我们也只考虑 $I(u) \cap I(v)$ 中的商品对应的评分值所对应的方差。注意, 根据该公式, 所得的 Pearson 相关系数并不等同于将原始的评价值利用平均值进行标准化再计算余弦相似度。

注意, 在这里计算 \bar{r}_u 和 \bar{r}_v 时有两种选择。我们考虑了用户 u 和用户 v 各自所有已知的评价情况, 并使用前面讨论过的公式

$$\bar{r}_u = \frac{1}{|I(u)|} \sum_{i \in I(u)} r_{ui}, \quad \bar{r}_v = \frac{1}{|I(v)|} \sum_{i \in I(v)} r_{vi} \quad (7-72)$$

来计算。这种方法的好处是对于每个用户来说, 只需要计算一次均值即可。

严格地讲, 这里我们是根据用户 u 和用户 v 都评价过的商品中的评价信息来计算 u 和 v 之间的相似度。因此, 在一些文献中也提出严格按照 Pearson 相关系数的计算方法, 只用 $I(u) \cap I(v)$ 中的商品来计算 \bar{r}_u 和 \bar{r}_v :

$$\bar{r}_u = \frac{1}{|I(u) \cap I(v)|} \sum_{i \in I(u) \cap I(v)} r_{ui}, \quad \bar{r}_v = \frac{1}{|I(u) \cap I(v)|} \sum_{i \in I(u) \cap I(v)} r_{vi} \quad (7-73)$$

在实际的比较中, 没有明确的证据证明其中一种方法优于另一种方法。若采用后一种计

算方法, 在计算每对用户 u 和 v 时, 都要重新计算 \bar{r}_u 和 \bar{r}_v 。因此, 在实际中, 出于降低计算复杂度的考虑, 通常采用前一种方法, 使用用户 u 的所有评价信息来计算其评价的平均值 \bar{r}_u 。

类似地, 也可以使用 Pearson 相关系数来计算商品之间的相似度:

$$PC(i, j) = \frac{\sum_{u \in U(i) \cap U(j)} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U(i) \cap U(j)} (r_{ui} - \bar{r}_i)^2 \sum_{u \in U(i) \cap U(j)} (r_{uj} - \bar{r}_j)^2}} \quad (7-74)$$

在式 (7-74) 中, 我们使用 $\bar{r}_i = \frac{1}{|U(i)|} \sum_{u \in U(i)} r_{ui}$ 作为均值来修正 r_{ui} 。在实际中, 人们发现使用用户的均值 \bar{r}_u 来修正 r_{ui} 远比使用 \bar{r}_i 来修正 r_{ui} 对于推荐更有效。主要原因在于不同的用户评分习惯差异较大, 使得使用用户的均值 \bar{r}_u 来修正 r_{ui} 更有效。因此, 在实际中, 人们又提出了修正的余弦相似度 (adjusted cosine similarity)。与前面 $PC(i, j)$ 的区别在于将 \bar{r}_i 替换为 \bar{r}_u :

$$AC(i, j) = \frac{\sum_{u \in U(i) \cap U(j)} (r_{ui} - \bar{r}_u)(r_{uj} - \bar{r}_u)}{\sqrt{\sum_{u \in U(i) \cap U(j)} (r_{ui} - \bar{r}_u)^2 \sum_{u \in U(i) \cap U(j)} (r_{uj} - \bar{r}_u)^2}} \quad (7-75)$$

在实际中计算商品间相似度时, 使用 $AC(i, j)$ 通常比 $PC(i, j)$ 更有效。

4. Spearman 秩相关系数

在计算 Pearson 相关系数时, 我们直接使用了评价值 r_{ui} 并利用平均值进行修正。在 Spearman 秩相关系数中, 我们先将评价值 r_{ui} 转化为它对应的排序。举一个简单的例子, 如果用户对用户 u , 如果他/她给 3 件商品的评分分别为 4、3、5, 那么评分对应的排序为 2、1、3 (从小到大的排列)。我们首先将用户 u 所有的评价 r_{ui} 转化为对应的排序, 并记为 k_{ui} , 并将所有 k_{ui} 的平均值记为 \bar{k}_u 。之后, 我们可以使用下面的式 (7-76) 来计算 Spearman 秩相关系数 (Spearman rank correlation, SRC):

$$SRC(u, v) = \frac{\sum_{i \in I(u) \cap I(v)} (k_{ui} - \bar{k}_u)(k_{vi} - \bar{k}_v)}{\sqrt{\sum_{i \in I(u) \cap I(v)} (k_{ui} - \bar{k}_u)^2 \sum_{i \in I(u) \cap I(v)} (k_{vi} - \bar{k}_v)^2}} \quad (7-76)$$

注意, 在计算中我们也只考虑了 $I(u) \cap I(v)$ 中的商品。在使用 Spearman 秩相关系数时, 我们不需要考虑评分值的标准化问题, 因为我们已经预先将其转化为排序了。但是, 当评价值只取有限的几个值时 (例如, 只能取 1 或者 2), 在将 r_{ui} 转化为 k_{ui} 的过程中, 很多评价值都会取相同的排序值 k_{ui} , 此时计算 Spearman 秩相关系数的意义不大。

5. 相似度的显著性

在计算相似度时, 假设我们有 5 个商品, 并假设用户如果购买过该商品, 则 $r_{ui} = 1$, 否则 $r_{ui} = 0$ 。考虑如下 4 个用户:

$$u_1 = [1, 0, 0, 0, 0], \quad u_2 = [1, 0, 0, 0, 0]$$

和

$$v_1 = [1, 1, 1, 0, 0], v_2 = [1, 1, 1, 0, 0]$$

假设都采用余弦相似度，则有：

$$CS(u_1, u_2) = CS(v_1, v_2) = 1.0$$

虽然这两对用户的相似度都为 1.0，但是直觉告诉我们 v_1 与 v_2 应该更相似，因为我们有更多的证据证明他们购买了更多相同的商品。

因此，在计算相似度的时候，我们要考虑相似度的显著性（significance），就是相似度计算中证据的强弱。其基本原理：如果相似度的计算只是基于较少的评价值，则我们要降低相似度的绝对值。在前面所讨论的多种计算用户 u 和用户 v 之间相似度的方法中，基本上都依赖于用户共同评价过的商品的数目，即 $|I(u) \cap I(v)|$ 。一种简单的方法是根据 $|I(u) \cap I(v)|$ 的大小对相似度进行修正：如果 $|I(u) \cap I(v)|$ 大于或等于某一阈值 γ ，对相似度不修正；如果 $|I(u) \cap I(v)|$ 小于阈值 γ ，要对算出的相似度进行“惩罚”，通常是降低其相似度的值。这种方法称为显著性权值（significance weighting）。在计算用户之间的相似度时，通常采用的修正公式为：

$$w'_{uv} = \frac{\min\{|I(u) \cap I(v)|, \gamma\}}{\gamma} w_{uv} \quad (7-77)$$

这里 $\gamma > 0$ ， w_{uv} 是原始的相似度。这里，如果相似度 w_{uv} 的计算所基于的共同评价个数 $|I(u) \cap I(v)|$ 小于 γ ，则要乘以系数 $|I(u) \cap I(v)| / \gamma$ 。

类似地，计算商品间相似度时，可以用下面的式（7-78）来修正：

$$w'_{ij} = \frac{\min\{|U(i) \cap U(j)|, \gamma\}}{\gamma} w_{ij} \quad (7-78)$$

在实际中，一般要使用交叉检验来得到最优的 γ 值。

6. 相似度计算的实例

这里我们用一个实例来说明如何计算上面讨论的那些相似度。

例 7-3 在这个简化的例子中，假设有 6 件商品，用户 U1 和用户 U2 对这 6 件商品的评分见表 7-2。

表 7-2 用于相似度计算的实例

	商品 I1	商品 I2	商品 I3	商品 I4	商品 I5	商品 I6
用户 U1		2	5	4		3
用户 U2	5	4	4	3		

在这个例子中，用户 U1 对商品 2、3、4 和 6 进行了评价；用户 U2 对商品 1、2、3 和 4

进行了评价。

(1) 余弦相似度。构建用户 U1 和用户 U2 对应的向量如下:

$$\mathbf{x}_1 = [0, 2, 5, 4, 0, 3]^T, \mathbf{x}_2 = [5, 4, 4, 3, 0, 0]^T$$

则他们之间的余弦相似度为:

$$CS(U1, U2) = \frac{0 \times 5 + 2 \times 4 + 5 \times 4 + 4 \times 3 + 0 \times 0 + 3 \times 0}{\sqrt{2^2 + 5^2 + 4^2 + 3^2} \sqrt{5^2 + 4^2 + 4^2 + 3^2}} = 0.67$$

(2) Jaccard 相似度。根据表 7-2, 有:

$$I(U1) = \{I2, I3, I4, I6\}, I(U2) = \{I1, I2, I3, I4\}$$

因此, 有:

$$I(U1) \cap I(U2) = \{I2, I3, I4\}, I(U1) \cup I(U2) = \{I1, I2, I3, I4, I6\}$$

直接使用 Jaccard 相似度的计算公式, 有:

$$J(U1, U2) = \frac{|I(U1) \cap I(U2)|}{|I(U1) \cup I(U2)|} = \frac{3}{5} = 0.6$$

(3) Pearson 相关系数。首先计算用户 U1 和 U2 评分的平均值:

$$\bar{r}_{U1} = \frac{2+5+4+3}{4} = 3.5, \bar{r}_{U2} = \frac{5+4+4+3}{4} = 4$$

在计算 Pearson 相关系数时, 我们只考虑用户 U1 和用户 U2 都评价过的商品。因此, 这里只需要考虑商品 I2、I3 和 I4。带入前面的计算公式中, 有:

$$\begin{aligned} PC(U1, U2) &= \frac{(2-3.5) \times (4-4) + (5-3.5) \times (4-4) + (4-3.5) \times (3-4)}{\sqrt{(2-3.5)^2 + (5-3.5)^2 + (4-3.5)^2} \sqrt{(4-4)^2 + (4-4)^2 + (3-4)^2}} \\ &= \frac{-0.5}{2.179449 \times 1} = -0.229 \end{aligned}$$

(4) Spearman 秩相关系数。首先我们将原始的评分转化为排序。注意, 用户 U2 的评分排好之后为 3、4、4、5, 对应的排序位置为 1、2.5、2.5、4。由于有两个相同的 4 分, 因此我们赋予 2.5 作为新的评分。转化后的新评分见表 7-3, 则对于用户 U1 和用户 U2, 其对应的新的评分的平均值分别为 $\bar{k}_{U1} = 2.5$ 和 $\bar{k}_{U2} = 2.5$ 。最后的 Spearman 秩相关系数为:

$$\begin{aligned} SRC(u, v) &= \frac{(1-2.5) \times (2.5-2.5) + (4-2.5) \times (2.5-2.5) + (3-2.5) \times (1-2.5)}{\sqrt{(1-2.5)^2 + (4-2.5)^2 + (3-2.5)^2} \sqrt{(2.5-2.5)^2 + (2.5-2.5)^2 + (1-2.5)^2}} \\ &= \frac{-0.75}{\sqrt{4.75} \sqrt{2.25}} = -0.229 \end{aligned}$$

表 7-3 转化为排序的新评分

	商品 I1	商品 I2	商品 I3	商品 I4	商品 I5	商品 I6
用户 U1		1	4	3		2
用户 U2	4	2.5	2.5	1		

从这个简单的例子可以看出,采用不同的相似度计算方法,就会取得不同的用户-用户相似度值,从而影响后面的用户-商品评分预测。因此,在实际中,我们需要根据数据的具体特征,选取最合适的计算方法。

7.5 R 中 recommenderlab 的实际使用

目前在 R 中关于推荐算法的软件包还较少,在本节我们介绍其中比较流行的 recommenderlab 软件包^①。该软件包实现了我们前面讨论的几类基本算法:

- 基于奇异值分解的推荐算法;
- 基于用户的邻域推荐算法;
- 基于商品的邻域推荐算法。

此外,该软件包还实现了若干简单的推荐算法,如为每个用户推荐最流行的商品等。使用该软件包,用户可以直接调用相关的算法,为每个用户推荐他/她最喜欢的商品或者直接预测特定用户-商品的评价值。

该软件包的缺点是实现的算法有限,同时不能处理大数据。例如,我们在前面介绍的很多复杂的矩阵分解算法都没有提供。我们在这里介绍 recommenderlab 包的主要目的是为了让学生在 R 中直接使用前面介绍的一些基本推荐算法,获得使用推荐算法的第一手经验。

下面我们首先讨论如何在 recommenderlab 包中保存和操作评价矩阵;之后介绍如何构建和使用推荐模型;在此基础上,我们讨论该包中的 evaluationScheme() 函数和 calcPredictionAccuracy() 函数,介绍如何划分评价矩阵和如何比较多个不同的推荐模型。更完全的关于该软件包的介绍可以参见网络上的相关资料^②。

在 recommenderlab 包中,评价矩阵是用一个抽象的 ratingMatrix 类来表示的。该抽象类定义了很多矩阵方面操作的方法。在实际中,我们使用它的两个具体的例子来保存评价矩阵:

- binaryRatingMatrix;
- realRatingMatrix。

在 binaryRatingMatrix 中,我们只能保存布尔型的评价值,而 realRatingMatrix 则

① <https://github.com/cran/recommenderlab>

② 手册: <https://cran.r-project.org/web/packages/recommenderlab/recommenderlab.pdf>。简介: <https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf>。

保存实数型的评价值。在实现上，它们都利用了 R 中相应的稀疏矩阵来保存和操作评价矩阵。

在使用 recommenderlab 包中的 `binaryRatingMatrix` 和 `realRatingMatrix` 时，我们可以使用如下的方法来探索评价矩阵。

- `dim()`：返回评价矩阵的行数和列数。
- `nrow()` 和 `ncol()`：分别返回评价矩阵的行数和列数（即用户数目和商品数目）。
- `nratings()`：返回评价矩阵中所有已知评价值的总数。
- `dimnames()`：返回评价矩阵中每行对应的用户名列表和每列对应的商品名列表。
- `rowCounts()` 和 `colCounts()`：分别计算每行和每列中已知的评价值的数目。
- `rowMeans()` 和 `colMeans()`：分别计算每行和每列中已知评价值的平均值。
- `rowSums()` 和 `colSums()`：分别计算每行和每列中已知评价值的和。

在后面的示例代码中，我们将利用具体的例子讲解如何使用这些函数来探索评价矩阵。

对于一个评价矩阵，可以使用 `as()` 函数将 `binaryRatingMatrix` 和 `realRatingMatrix` 转化为列表、矩阵和数据框。假设 `RM` 是一个 recommenderlab 中的评价矩阵，则可以使用如下代码来将其进行类型转换：

```
RM_list <- as(RM, 'list')
RM_matrix <- as(RM, 'matrix')
RM_df <- as(RM, 'data.frame')
```

对于评价矩阵，可以利用 recommenderlab 中的 `normalize()` 来进行标准化处理。在使用 `normalize()` 函数时，我们主要有两个重要参数，即 `method` 和 `row`，下面是两种不同的用法：

```
normalize(RM, method='center', row=T)
normalize(RM, method='Z-score', row=F)
```

这里 `RM` 是一个评价矩阵，参数 `method` 表示不同的预处理方法，其中 `'center'` 表示对每个用户或者商品的评价值进行预处理使得其平均值是 0，而 `'Z-score'` 表示标准化处理后每个用户或者商品的评价值的平均值是 0 且标准差是 1；参数 `row` 的取值为 `T` 或者 `F`，分别表示对每个用户（即每行）或者每个商品（即每列）分别做预处理。

我们使用 recommenderlab 的核心任务是构建一个推荐模型，以及利用推荐模型来预测推荐结果。利用提供的 `Recommender()` 函数，可以训练不同的推荐模型，其用法如下：

```
Recommender(RM, method, parameter = NULL)
```

这里的 `RM` 是一个用来训练模型的评价矩阵，可以使用参数 `method` 和 `parameter` 来进一步指定所要构建的模型。这里 `method` 用来指定模型的类型，常用的模型类型有以下几个。

- **RANDOM**：随机推荐模型。
- **POPULAR**：利用训练数据得到最流行的 k 个商品，并将它们推荐给所有用户。
- **IBCF**：基于商品的协同过滤（item-based collaborative filtering），即基于商品的邻域推荐算法。

- UBCF: 基于用户的协同过滤 (user-based collaborative filtering), 即基于用户的邻域推荐算法。
- SVD 和 SVDF: 基于矩阵分解的推荐模型。

这里参数 `parameter` 用来指定每个模型的具体控制参数。

在使用 `Recommender()` 函数构建模型时, 我们需要知道当前所有可以使用的参数选择。我们可以使用

```
recommenderRegistry$get_entries()
```

得到 `recommenderlab` 包当前提供的所有可以使用的参数设置。进一步, 我们可以使用

```
recommenderRegistry$get_entries(dataType = "realRatingMatrix")
```

得到所有适用于 `realRatingMatrix` 的设置; 可以使用

```
recommenderRegistry$get_entries(dataType = "binaryRatingMatrix")
```

得到所有适用于 `binaryRatingMatrix` 的设置。这里我们首先列出对于 `realRatingMatrix` 适用的所有参数设置:

```
$IBCF_realRatingMatrix
Recommender method: IBCF
Description: Recommender based on item-based collaborative filtering (real data).
Parameters:
  k method normalize normalize_sim_matrix alpha na_as_zero
1 30 Cosine      center                FALSE 0.5      FALSE
```

```
$POPULAR_realRatingMatrix
Recommender method: POPULAR
Description: Recommender based on item popularity (real data).
Parameters:
  normalize aggregationRatings aggregationPopularity
1 center      <function>                <function>
```

```
$RANDOM_realRatingMatrix
Recommender method: RANDOM
Description: Produce random recommendations (real ratings).
Parameters: None
```

```
$RERECOMMEND_realRatingMatrix
Recommender method: RERECOMMEND
Description: Re-recommends highly rated items (real ratings).
Parameters:
```

```
  randomize minRating
1 1 1 NA
```

```
$SVD_realRatingMatrix
Recommender method: SVD
```

Description: Recommender based on SVD approximation with column-mean imputation (real data).

Parameters:

```
k maxiter normalize
1 10      100      center
```

\$SVDF_realRatingMatrix

Recommender method: SVDF

Description: Recommender based on Funk SVD with gradient descend (real data).

Parameters:

```
k gamma lambda min_epochs max_epochs min_improvement normalize
1 10 0.015 0.001          50          200          1e-06      center
verbose
1 FALSE
```

\$UBCF_realRatingMatrix

Recommender method: UBCF

Description: Recommender based on user-based collaborative filtering (real data).

Parameters:

```
method nn sample normalize
1 cosine 25 FALSE      center
```

这里我们简单解释一下上面列出的参数设置情况。以基于用户的邻域算法为例，其参数设置对应于上面的\$UBCF_realRatingMatrix。在使用时，有 4 个参数可以设置：

- method
- nn
- sample
- normalize

这里参数 method 是计算相似度的方法，对于 realRatingMatrix，在 recommenderlab 包中提供了 'pearson'、'cosine' 和 'jaccard'。参数 nn 表示邻域大小。参数 sample 值是 T 或者 F，表示是否对训练集进行取样来训练推荐模型。参数 normalize 表示对评价值进行预处理的方法，常用的有 'center' 和 'z-score'，其含义与我们前面讨论 normalize() 函数时相同。

对于 binaryRatingMatrix，所有适用的参数设置包括：

\$AR_binaryRatingMatrix

Recommender method: AR

Description: Recommender based on association rules.

Parameters:

```
support confidence maxlen      measure verbose decreasing
1      0.1          0.3      2 confidence FALSE      TRUE
```

\$IBCF_binaryRatingMatrix

Recommender method: IBCF

Description: Recommender based on item-based collaborative filtering (binary rating data).

Parameters:

k method normalize_sim_matrix alpha
1 30 Jaccard FALSE 0.5

\$POPULAR_binaryRatingMatrix

Recommender method: POPULAR

Description: Recommender based on item popularity (binary data).

Parameters: None

\$RANDOM_binaryRatingMatrix

Recommender method: RANDOM

Description: Produce random recommendations (binary ratings).

Parameters: None

\$UBCF_binaryRatingMatrix

Recommender method: UBCF

Description: Recommender based on user-based collaborative filtering (binary data).

Parameters:

method nn weighted sample
1 jaccard 25 TRUE FALSE

同样地,我们仍以基于用户的邻域算法为例说明对于 `binaryRatingMatrix` 的参数设置。其参数设置对应于上面的 `$UBCF_binaryRatingMatrix`。在使用时,有4个参数可以设置:

- method
- nn
- weighted
- sample

这里参数 `method` 是计算相似度的方法;参数 `nn` 表示邻域大小;参数 `sample` 值是 T 或者 F,表示是否对训练集进行取样来训练推荐模型。与前面对于 `realRatingMatrix` 的参数设置不同的是,这里的参数 `weighted` 表示在计算预测值时是否考虑其邻域中不同用户的权值。

在构建推荐模型后,可以使用 `predict()` 函数为用户进行推荐。使用 `predict()` 有两种模式:

- 为每个用户推荐 `topN` 个最喜欢的商品;
- 为指定的用户-商品对预测评价价值。

该函数的用法如下:

```
predict(M, newdata, n=10, type="topNList",...).
```

这里 `M` 是前面使用 `Recommender()` 函数返回的对象(即已经构建的推荐模型),`newdata` 则指定了新的用户及其历史评价数据。这里 `newdata` 也可以是训练 `M` 时所用的评价矩阵所对应的用户的一部分,此时我们可直接指定 `newdata` 为训练用的评价矩阵中用户对应的索引。参数 `type` 可以设置为 `"topNList"`、`"ratings"`、`"ratingMatrix"` 之一,用来指定不同的预测任务。其

中"topNList"表示返回 newdata 中每个用户最喜好的前 n (对应参数 n) 个商品; 当 type 为 "ratings" 或 "ratingMatrix" 时, 参数 n 被忽略, predict() 函数预测用户-商品的评价值, 并返回一个 realRatingMatrix 对象, 其中每行对应一个用户。这里 "ratings" 和 "ratingMatrix" 的区别是使用 ratingMatrix 时我们将返回的 realRatingMatrix 中所有 newdata 中的已知用户-商品评价对的评价值直接设为 newdata 中的真实评价值。

与前面的分类或者回归问题类似, 需要对所得模型在测试集上的效果进行评价。在 recommenderlab 包中, 我们利用所提供的 evaluationScheme() 和 calcPredictionAccuracy() 函数来比较多个推荐模型的性能。利用 evaluationScheme() 函数, 可将数据分为 3 部分:

- 训练数据;
- 测试数据中的已知数据;
- 测试数据中的未知数据。

在划分数据时, 我们将用户分为两部分: 训练集中的用户和测试集中的用户。对于测试集中的用户对应的评价值, 进一步分为两部分: 已知数据和未知数据。其中训练数据是用来训练推荐模型的。而为测试集中的用户推荐商品时, 我们利用已经训练好的模型和测试数据中的已知数据来进行推荐, 并将推荐结果与未知部分进行比较从而知道所得模型的好坏。在下面的表 7-4~表 7-7 中, 我们使用一个具体的例子来说明如何使用 evaluationScheme() 来划分评价矩阵。在这个例子中, 我们有 4 个用户和 4 件商品, 将用户 User1 和 User2 划入训练数据, 将 User3 和 User4 划入测试数据。表 7-4 是原始的评价矩阵, 表 7-5 是划分所得的训练数据, 表 7-6 是测试数据中的已知部分, 表 7-7 是测试数据的未知部分。在这个例子中, 在测试数据的未知部分中, 每个用户只有一件商品。

表 7-4 原始的评价矩阵

Rating matrix	Item1	Item2	Item3	Item4
User1	4	5		1
User2	3	4	1	
User3	1		3	5
User4	4	5		

表 7-5 训练数据

Rating matrix	Item1	Item2	Item3	Item4
User1	4	5		1
User2	3	4	1	

表 7-6 测试数据中的已知部分

Rating matrix	Item1	Item2	Item3	Item4
User3	1			5
User4		5		

表 7-7 测试数据中的未知部分

Rating matrix	Item1	Item2	Item3	Item4
User3			3	
User4	4			

在具体使用 `evaluationScheme()` 函数时, 其用法如下:

```
evaluationScheme(data, method="split", train=0.9, k=10, given=3)
```

首先我们用 `data` 来指定评价矩阵, 然后用参数 `method` 来指定不同的划分方法, `recommenderlab` 中提供的选择包括以下几个。

- 'split': 简单的划分。
- 'bootstrap': bootstrap 取样。
- 'cross-validation': 交叉检验。

参数 `train` 表示其中训练集的比例, 参数 `k` 表示交叉检验中的重数, 参数 `given` 表示在测试集中已知数据部分对于每个用户包含多少个评价。注意, 这里 `given` 可以取负值, 例如, `given=-10` 表示对于每个用户, 我们对测试集中的用户选择 10 个评价作为测试数据中的未知数据。在表 7-4 至表 7-7 给出的例子中我们将 `given` 设为 -1。

在得到 `evaluationScheme()` 函数生成的结果后, 可以使用 `getData()` 函数得到所划分的 3 组数据。下面我们使用 `recommenderlab` 包自带的 `MovieLense` 数据来说明:

```
ML_e <- evaluationScheme(MovieLense,
                          method='split',
                          train=0.9,
                          given = -10,
                          goodRating=5)
ML_train <- getData(ML_e, 'train')
ML_test_known <- getData(ML_e, 'known')
ML_test_unknown <- getData(ML_e, 'unknown')
```

在这个例子中, 首先使用 `evaluationScheme()` 函数生成一个划分 `ML_e`, 然后使用 `getData()` 函数和不同的参数 ('train'、'known'、'unknown') 得到所划分的 3 个数据集。注意, 这里我们还设置了参数 `goodRating`。在 `MovieLense` 数据集中, 评价值的取值为 1~5。这里, `goodRating=5` (表示评价值为 5) 或者以上的部分我们认为是正类, 其他则视为负类, 这些设置在使用 `calcPredictionAccuracy()` 函数评估模型性能时会用到。

对于推荐模型的评价, 可以使用 `recommenderlab` 包中提供的 `calcPredictionAccuracy()` 函数来计算。对于评价值的预测 (即调用 `predict()` 函数时将 `type` 设为 'ratings' 或者 'ratingMatrix'), 使用 `calcPredictionAccuracy()` 函数可以得到:

- 平均误差 (mean average error, MAE);
- 均方差 (mean squared error, MSE);
- 均方根误差 (root mean squared error, RMSE)。

当预测结果是每个用户对应的 topN 件商品时,使用 `calcPredictionAccuracy()` 函数可以得到 TP、FP、FN、TN、TPR、FPR,以及召回率和精确率。这里我们需要有“正类”和“负类”的概念,这也是对 `realRatingMatrix` 类型的评价矩阵使用 `evaluationScheme()` 函数时需要指定 `goodRating` 的原因。在使用 `calcPredictionAccuracy()` 函数时,第一个输入参数对应模型的预测结果,第二个输入参数对应真实数据。

下面我们用具体的示例代码来说明使用 `recommenderlab` 包来探索数据、建立模型,以及评价、比较模型的全过程。完全的 R 代码在文件 `recommenderlab_example.R` 中。

在代码中,首先检查 `recommenderlab` 包是否已经安装;如果没有,则使用 `install.packages()` 函数安装该软件包。

接下来载入 `recommenderlab` 包自带的 3 个数据集: `MovieLense`、`Jester5k` 和 `MSWeb`。每个数据集对应一个同名的评价矩阵对象。其中 `MovieLense` 和 `Jester5k` 是 `realRatingMatrix`,而 `MSWeb` 是 `binaryRatingMatrix`。

```
data(MovieLense)
data(Jester5k)
data(MSWeb)
```

然后,探索数据的评价矩阵,代码如下:

```
# Visualizing a sample of this
image(sample(MovieLense, 500), main = "Raw ratings")
image(MSWeb[1:500,], main='raw ratings')

summary(getRatings(MovieLense))
# check the size of MovieLense directly
n_user_num_movie <- dim(MovieLense)[1]
n_item_num_movie <- dim(MovieLense)[2]
# Other functions we can use to explore rating matrix dimensions
user_list_movie <- dimnames(MovieLense)[1]
item_list_movie <- dimnames(MovieLense)[2]
# Get all ratings as a numeric vector
Movie_v <- getRatings(MovieLense)
hist(Movie_v, main='Histogram of ratings', xlab='Rating')
# Get all ratings as a sparse matrix
Movie_Msp <- getRatingMatrix(MovieLense)
# use the list of functions provided by the package
# we can print out the class information
str(Movie_v)
str(Movie_Msp)
# Next we compute some statistics
Movie_n_ratings <- nratings(MovieLense)
hist(rowCounts(MovieLense), breaks=50, xlab="Num of users", ylab="Num of movies rated")
# Explore the mean rating for each movie
hist(colMeans(MovieLense), breaks=50, xlab="Rating", ylab="Num of movies")
```


在这一步首先使用 `image()` 函数简单地可视化 MovieLens 和 MSWeb 两个评价矩阵。这里, 使用 `sample()` 函数从 MovieLens 中随机选择 500 个用户对应的评价数据进行可视化。可视化结果分别如图 7-2 和图 7-3 所示。从图 7-2 和图 7-3 中可以看出, MSWeb 比 MovieLens 要稀疏很多。

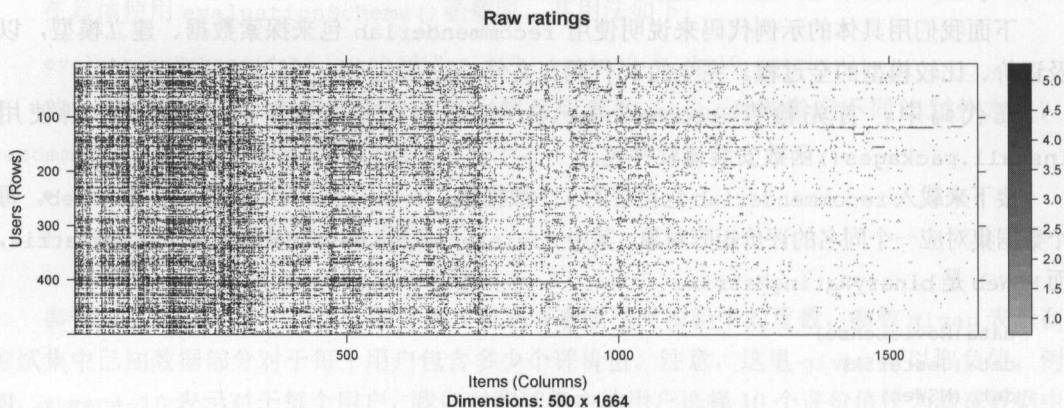


图 7-2 MovieLens 中的评价矩阵

我们使用 `dim()` 函数检查 MovieLens 数据集中用户数目和商品数目, 使用 `dimnames()` 函数得到了所有的用户名和商品名。这里我们着重使用了如下几个函数来从 MovieLens 中得到评价值。

- `getRatings()` 函数: 将评价矩阵中所有已知的评价值以向量的形式返回。
- `getRatingMatrix()` 函数: 将评价矩阵中所有已知的评价值以矩阵的形式返回, 其行数和列数与输入的评价矩阵相同。
- `nratings()` 函数: 返回评价矩阵中所有已知评价值的数目, 即 `nratings(R)` 等价于 `length(getRatings(R))`, 这里 `R` 是输入的评价矩阵。

上面一段程序的输出为:

```
> summary(getRatings(MovieLens))
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.00   3.00   4.00   3.53   4.00   5.00

> str(Movie_v)
num [1:99392] 5 4 4 4 4 3 1 5 4 5 ...

> str(Movie_Msp)
Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
..@ i      : int [1:99392] 0 1 4 5 9 12 14 15 16 17 ...
..@ p      : int [1:1665] 0 452 583 673 882 968 994 1386 1605 1904 ...
..@ Dim     : int [1:2] 943 1664
..@ Dimnames:List of 2
.. ..$ : chr [1:943] "1" "2" "3" "4" ...
```

```

... ..$ : chr [1:1664] "Toy Story (1995)" "GoldenEye (1995)" "Four Rooms (1995)" "Get Shorty (1995)" ...
..@ x      : num [1:99392] 5 4 4 4 4 3 1 5 4 5 ...
..@ factors : list()'

```

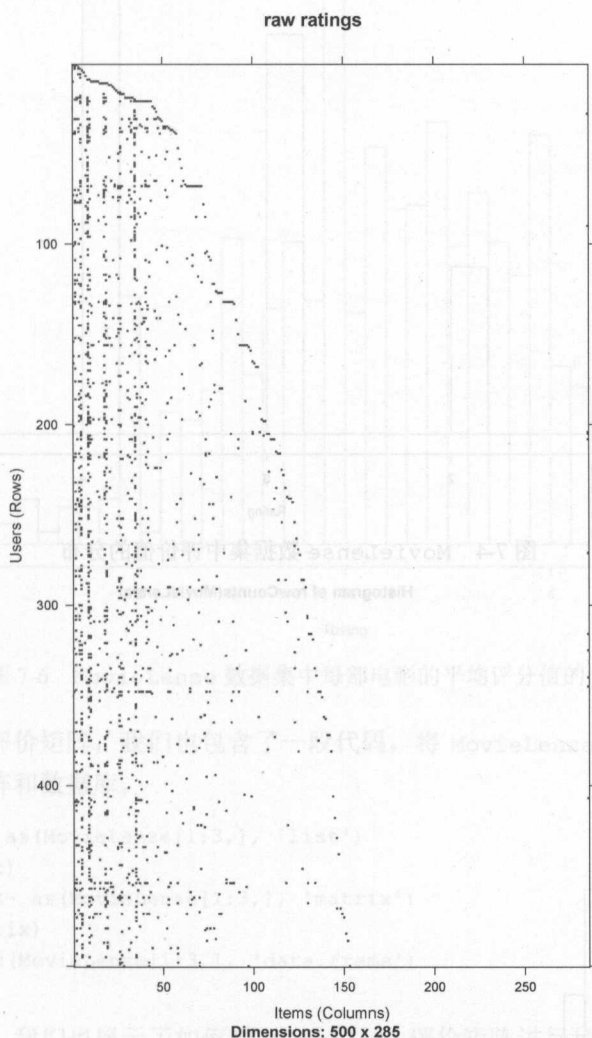


图 7-3 MSWeb 数据集中的评价矩阵

在上面的那段程序中，我们还使用 `hist()` 函数来探索评价矩阵的分布。首先使用 `hist(Movie_v, main='Histogram of ratings', xlab='Rating')` 来得到所有评价值的分布，如图 7-4 所示。从图 7-4 中可以看出，评价值为 1~5，其中得分为 4 的评价值最多。此外，我们还探索了每个用户评价过的电影数的分布和每部电影的平均评分值的分布，分别如图 7-5 和图 7-6 所示。

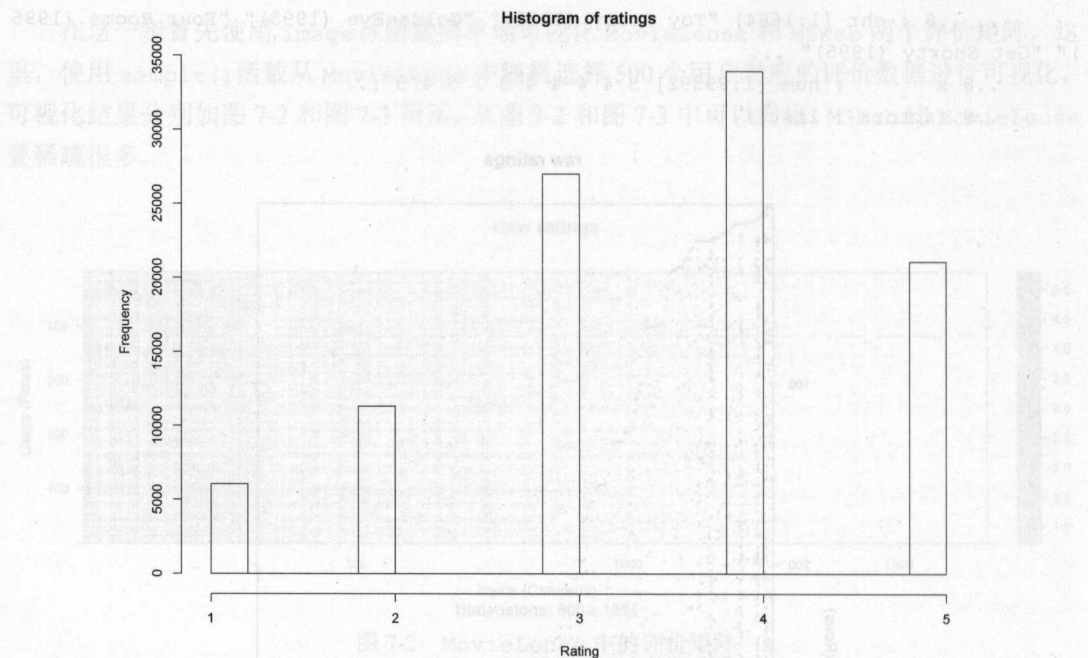


图 7-4 MovieLens 数据集中评价值的分布

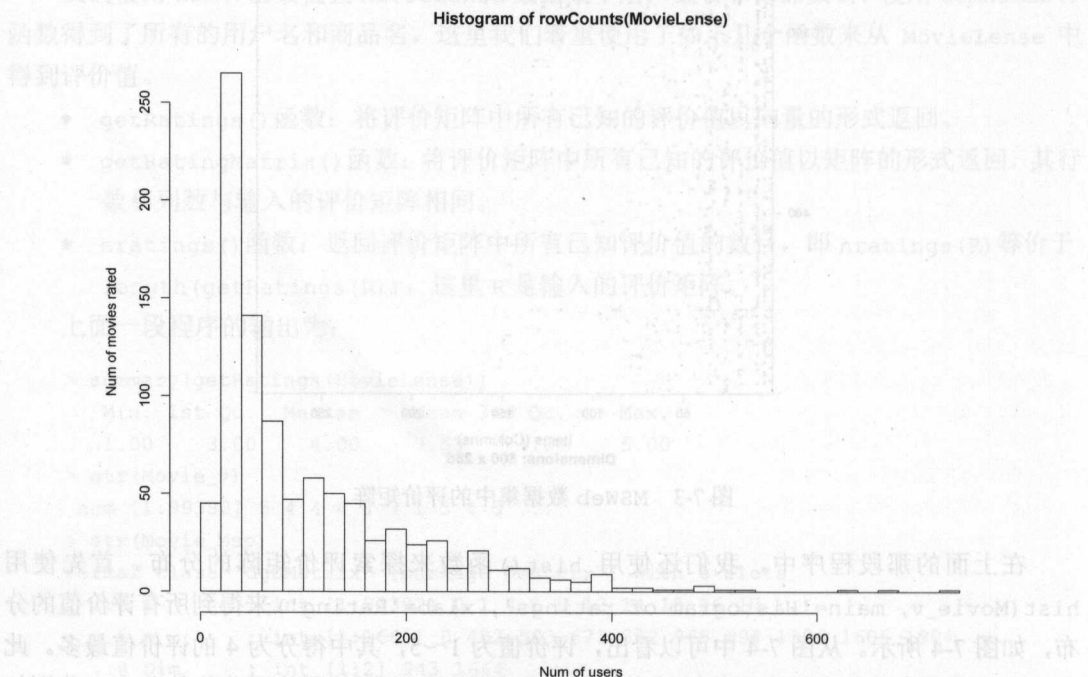


图 7-5 MovieLens 数据集中每个用户评价过的电影数的分布

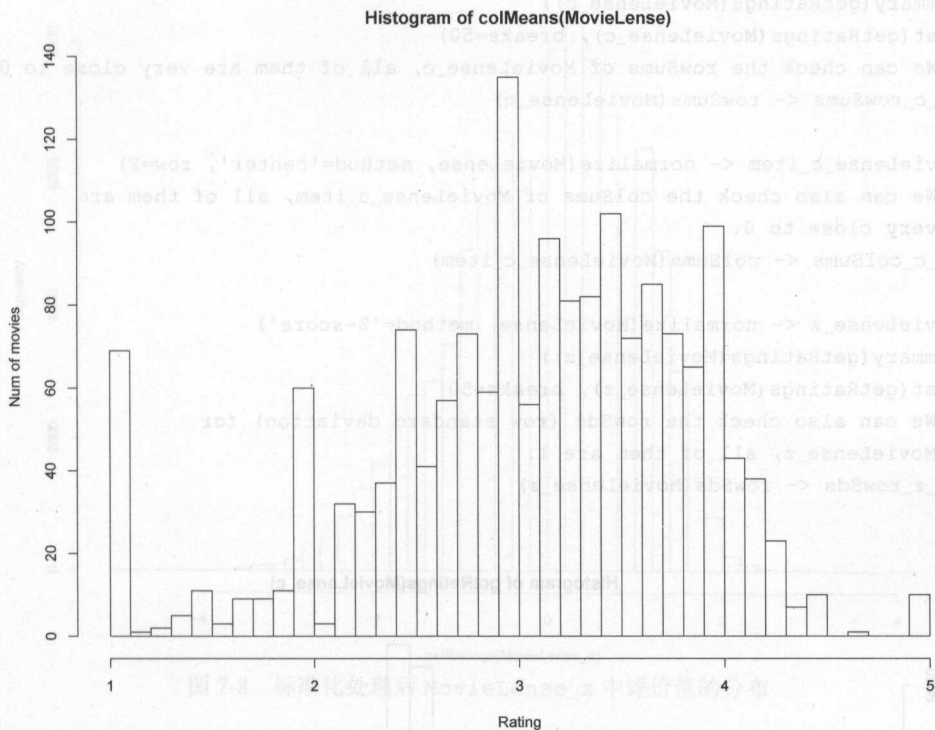


图 7-6 MovieLens 数据集中每部电影的 average 评分值的分布

为了方便操作评价矩阵，我们也包含了一段代码，将 MovieLens 评价矩阵的前 3 行分别转化为列表、矩阵和数据库。

```
ML_3u_list <- as(MovieLens[1:3,], 'list')
str(ML_3u_list)
ML_3u_matrix <- as(MovieLens[1:3,], 'matrix')
str(ML_3u_matrix)
ML_3u_df <- as(MovieLens[1:3,], 'data.frame')
str(ML_3u_df)
```

在示例代码中，我们也展示了如何对 MovieLens 评价矩阵进行预处理（代码如下）。其中 MovieLens_c 使得每个用户的平均评价值为 0（即对每个用户进行中心化）；MovieLens_c_item 使得每件商品的平均评价值为 0；MovieLens_z 使得每个用户的平均评价值为 0，评价值的标准差为 1（即对每个用户进行标准化）。这里我们还进一步计算了 ML_c_rowSums、ML_c_colSums、ML_z_rowSds 来进行验证。我们还探索了 MovieLens_c 和 MovieLens_z 中评价值的分布，分别如图 7-7 和图 7-8 所示。从图 7-7 和图 7-8 中可以看出，与原始的评价值的分布相比，预处理后的评价值分布更加接近于正态分布，而且 MovieLens_z 的分布更加接近一些。


```

MovieLense_c <- normalize(MovieLense, method='center')
summary(getRatings(MovieLense_c))
hist(getRatings(MovieLense_c), breaks=50)
# We can check the rowSums of MovieLense_c, all of them are very close to 0.
ML_c_rowSums <- rowSums(MovieLense_c)

MovieLense_c_item <- normalize(MovieLense, method='center', row=F)
# We can also check the colSums of MovieLense_c_item, all of them are
# very close to 0.
ML_c_colSums <- colSums(MovieLense_c_item)

MovieLense_z <- normalize(MovieLense, method='Z-score')
summary(getRatings(MovieLense_z))
hist(getRatings(MovieLense_z), breaks=50)
# We can also check the rowSds (row standard deviation) for
# MovieLense_z, all of them are 1.
ML_z_rowSds <- rowSds(MovieLense_z)

```

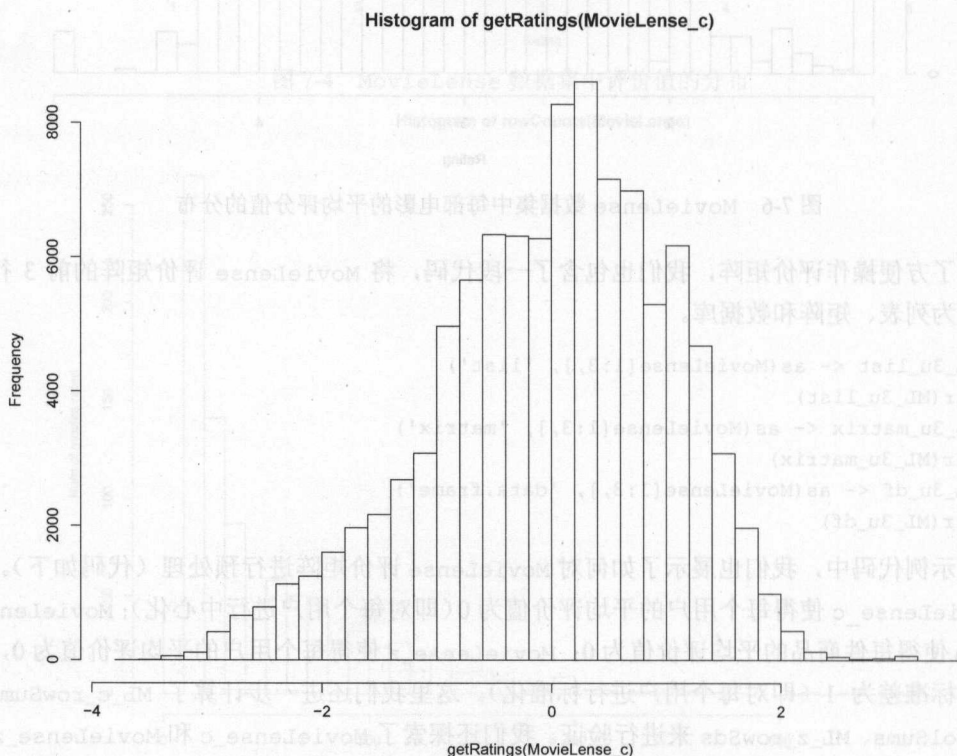


图 7-7 中心化后 MovieLense_c 中评价值的分布

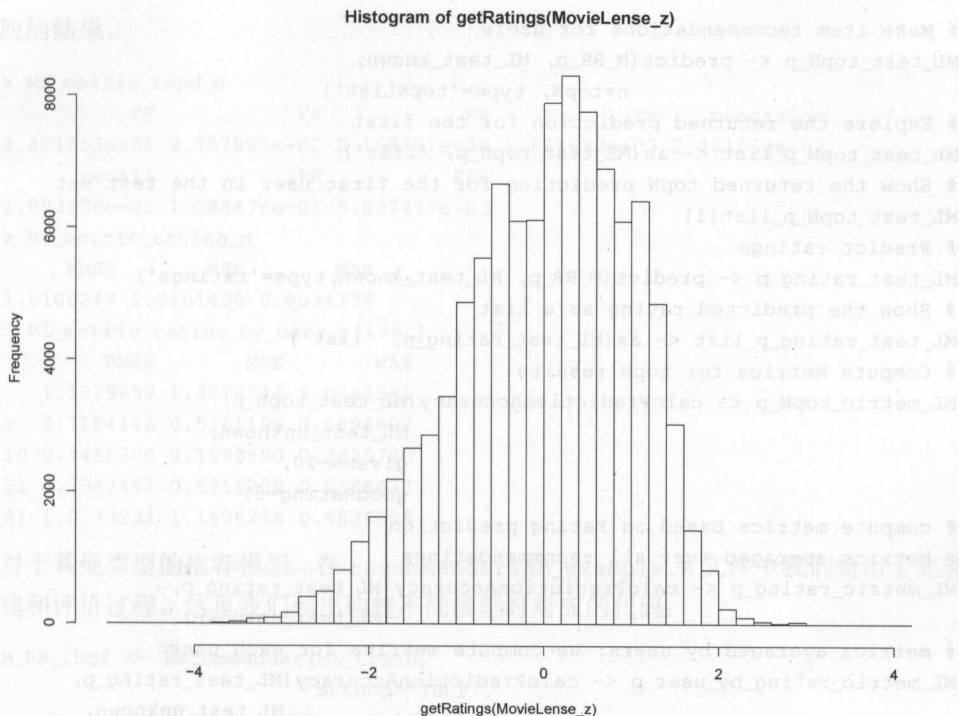


图 7-8 标准化处理后 MovieLense_z 中评价值的分布

下面讲解如何利用 MovieLense 数据构建不同的模型，并在测试集上进行预测，最后调用 `calcPredictionAccuracy()` 函数计算所得模型在测试集上的性能。首先使用 `evaluationScheme()` 函数将 MovieLense 数据分为 3 部分。

```
ML_e <- evaluationScheme(MovieLense,
                          method='split',
                          train=0.9,
                          given = -10,
                          goodRating=5)
ML_train <- getData(ML_e, 'train')
ML_test_known <- getData(ML_e, 'known')
ML_test_unknown <- getData(ML_e, 'unknown')
topN <- 10
```

这里我们就得到了 `ML_train`、`ML_test_known` 和 `ML_test_unknown`。在下面的例子中，我们都使用 `ML_train` 来训练推荐模型，再使用 `ML_test_known` 作为 `predict()` 函数的输入来得到推荐结果，最后比较模型的推荐结果和 `ML_test_unknown` 从而计算推荐模型的性能。

我们构建的第一个模型是 `recommenderlab` 中提供的一个很简单的 `POPULAR` 模型，即从训练集中找出最流行的 `topN` 件商品并推荐给测试集中所有的用户。对应的 R 代码如下：

```
M_RR_p <- Recommender(ML_train,
                      method='POPULAR')
```

```

# Make item recommendations for users
ML_test_topN_p <- predict(M_RR_p, ML_test_known,
                        n=topN, type='topNList')
# Explore the returned prediction for the first
ML_test_topN_p_list <- as(ML_test_topN_p, 'list')
# Show the returned topN prediction for the first user in the test set
ML_test_topN_p_list[1]
# Predict ratings
ML_test_rating_p <- predict(M_RR_p, ML_test_known, type='ratings')
# Show the predicted rating as a list
ML_test_rating_p_list <- as(ML_test_rating_p, 'list')
# Compute metrics for topN results
ML_metric_topN_p <- calcPredictionAccuracy(ML_test_topN_p,
                                          ML_test_unknown,
                                          given=-10,
                                          goodRating=5)

# compute metrics based on rating prediction
# metrics averaged over all recommendations
ML_metric_rating_p <- calcPredictionAccuracy(ML_test_rating_p,
                                          ML_test_unknown)
# metrics averaged by users: we compute metrics for each user
ML_metric_rating_by_user_p <- calcPredictionAccuracy(ML_test_rating_p,
                                                    ML_test_unknown,
                                                    byUser=T)

```

这里我们调用 `Recommender()` 函数来构建推荐模型。在构建 POPULAR 模型时,只需要将 `method` 参数设为 'POPULAR'。然后,调用 `predict()` 函数两次,分别将 `type` 参数设为 'topNList' 和 'ratings'。对于 topNList 的返回结果 `ML_test_topN_p`,可以将其转化为列表 `ML_test_topN_p_list`。这里我们使用 `ML_test_topN_p_list[1]` 来直接观察模型的返回结果。

```

> ML_test_topN_p_list[1]
$'7'
[1] "Titanic (1997)"          "Schindler's List (1993)"
[3] "L.A. Confidential (1997)" "Good Will Hunting (1997)"
[5] "Toy Story (1995)"        "Alien (1979)"
[7] "Close Shave, A (1995)"   "Wrong Trousers, The (1993)"
[9] "Apt Pupil (1998)"        "Lone Star (1996)"

```

类似地,也可以将 `ML_test_rating_p` 转化为列表对象 `ML_test_rating_p_list` 以方便查看。对于两种不同的预测结果 `ML_test_topN_p` 和 `ML_test_rating_p`,可以使用 `calcPredictionAccuracy()` 函数来评价模型的性能。在使用 `calcPredictionAccuracy()` 函数计算评价预测结果时,还有一个控制参数 `byUser`。该参数的默认值是 `F`,表示将所有的用户一起考虑;当其值是 `T` 时,表示单独为每个用户计算结果。在上面的示例程序中,我们为每个用户单独计算了 MAE、MSE 和 RMSE,并保存在 `ML_metric_rating_by_user_p` 中。我们可以直接在 R 的控制端查看如下结果,其中对于 `ML_metric_rating_by_user_p`,只查看了前 5 个用

户对应的结果。

```
> ML_metric_topN_p
      TP      FP      FN      TN  precision
2.421053e-01 9.757895e+00 2.168421e+00 1.661832e+03 2.421053e-02
      recall      TPR      FPR
1.083676e-01 1.083676e-01 5.837417e-03
> ML_metric_rating_p
      RMSE      MSE      MAE
1.0100247 1.0201499 0.8034736
> ML_metric_rating_by_user_p[1:5,]
      RMSE      MSE      MAE
7 1.1779959 1.3876744 1.0343581
9 0.7184146 0.5161196 0.5896807
10 0.4465300 0.1993890 0.3825720
21 0.7947457 0.6316208 0.6586477
31 1.0773234 1.1606256 0.8836956
```

对于其他类型的推荐模型，在 `recommenderlab_example.R` 文件中我们给出了完整的模型构建和评价过程。这里我们列出构建不同模型时对应的代码：

```
M_RR_ibcf <- Recommender(ML_train,
                          method='IBCF',
                          param=list(k=20,
                                     normalize='center'))
M_RR_ubcf <- Recommender(ML_train,
                          method='UBCF',
                          param=list(method='cosine',
                                     nn=30,
                                     normalize='center'))
M_RR_ubcf2 <- Recommender(ML_train,
                           method='UBCF',
                           param=list(method='pearson',
                                      nn=30,
                                      sample=F,
                                      normalize='center'))
M_RR_svd <- Recommender(ML_train,
                         method='SVD',
                         param=list(k=15,
                                    maxiter=500,
                                    normalize='center'))
M_RR_svdf <- Recommender(ML_train,
                          method='SVDF',
                          param=list(k=15,
                                     lambda=0.001,
                                     max_epochs=50,
                                     normalize='center'))
```

这里我们分别构建了一个基于商品的邻域方法模型 `M_RR_ibcf`，两个基于用户的邻域方

法模型 `M_RR_ubcf` 和 `M_RR_ubcf2`, 两个基于矩阵分解的模型 `M_RR_svd` 和 `M_RR_svdf`。在构建不同的模型时, 我们需要将参数 `method` 设为不同的值, 如 'IBCF'、'UBCF'、'SVD' 和 'SVDF' 等。对于每种方法, 我们再将它对应的参数用一个列表对象组织起来赋给 `param` 参数。在 IBCF 中, 我们使用参数 `k` 指定邻域的大小, 使用参数 `normalize` 指定采用何种方法对评价值进行预处理, 选项包括 'center' 和 'z-score'。在 UBCF 中, 我们使用 `nn` 指定邻域的大小, 使用 `method` 指定采用何种方法计算相似度, 参数 `sample` 表示是否对用于训练的评价矩阵进行取样, 参数 `normalize` 同样表示如何对评价值进行预处理。在 SVD 中, 参数 `k` 表示矩阵分解中用户和商品的低维表示的维数, 参数 `maxiter` 表示计算矩阵分解时算法的最多运行次数, `normalize` 表示对评价矩阵进行预处理的方法。算法 SVDF 与 SVD 类似, 区别在于 SVDF 中使用了梯度下降法。在 SVDF 中, 参数 `k` 表示低维表示的维数, `lambda` 表示正则化项的权重, `max_epochs` 表示梯度下降法运行的次数, `normalize` 也表示对评价矩阵进行预处理的方法。

上面的 MovieLense 是 `realRatingMatrix` 对象。接下来我们考虑 `binaryRatingMatrix` 类型的 MSWeb 数据。在 `recommenderlab_example.R` 文件中, 我们给出了关于 MSWeb 数据完整的模型构建和评价过程, 在这里只简单列出构建多个不同推荐模型的代码并进行说明。

```
M_BR_p <- Recommender(MSW_train,
                      method='POPULAR')
M_BR_ibcf <- Recommender(MSW_train,
                        method='IBCF',
                        param=list(k=20,
                                method='Jaccard',
                                normalize_sim_matrix=F))
M_BR_ubcf <- Recommender(MSW_train,
                        method='ubcf',
                        param=list(method='Jaccard',
                                nn = 20,
                                weighted=T))
```

可以看出, 其参数设置与 `realRatingMatrix` 大致相同, 只需要根据 `recommenderlab` 包的要求稍微调节一下具体参数即可。

之后我们调用 `recommenderlab` 中的 `HybridRecommender()` 函数将已有的多个推荐模型以线性组合的形式综合在一起。在使用该函数时, 需要指定要综合哪些推荐模型, 并在参数 `weights` 中指定它们的权重。在下面的例子中, 我们综合了 3 个不同的模型 `M_RR_ibcf`、`M_RR_ubcf` 和 `M_RR_svd`, 它们对应的权重分别为 0.3、0.2 和 0.5。得到综合后的模型后, 可以使用 `predict()` 函数根据综合后的模型进行预测。

```
M_RR_h1 <- HybridRecommender(M_RR_ibcf,
                             M_RR_ubcf,
                             M_RR_svd,
                             weights=c(0.3, 0.2, 0.5))
ML_test_topN_h1 <- predict(M_RR_h1, ML_test_known,
                          n=topN, type='topNList')
```

```
ML_test_rating_h1 <- predict(M_RR_h1, ML_test_known, type='ratings')
```

最后, 我们利用 recommenderlab 提供的 evaluate() 函数来比较不同的推荐模型。我们可以利用上面介绍的 Recommender()、predict() 和 calcPredictionAccuracy() 等诸函数来逐一构建并比较多个不同的模型。但是利用提供的 evaluate() 函数, 可以更加直接和方便地比较多个不同的模型。在下面的例子中, 我们使用 Jester5k 数据集中的前 1000 个用户的数据来训练和比较不同的模型。

```
J_e <- evaluationScheme(Jester5k[1:1000],
                        method="split",
                        train = .9,
                        k=1,
                        given=-5,
                        goodRating=5)
algorithms2compare <- list(
  "random items" = list(name="RANDOM", param=NULL),
  "popular items" = list(name="POPULAR", param=NULL),
  "user-based CF" = list(name="UBCF", param=list(nn=50)),
  "item-based CF" = list(name="IBCF", param=list(k=50)),
  "SVD approximation" = list(name="SVD", param=list(k = 50))
)
J_results_topN <- evaluate(J_e, algorithms2compare,
                          type = "topNList",
                          n=c(1, 3, 5, 10, 15, 20))
J_results_topN
names(J_results_topN)
J_results_topN[["user-based CF"]]
plot(J_results_topN, annotate=c(1,2,4), legend="bottomright")
plot(J_results_topN, "prec/rec", annotate=4, legend="topleft")
# Compare ratings
J_results_rating <- evaluate(J_e, algorithms2compare, type = "ratings")
J_results_rating
plot(J_results_rating)
```

在上面的代码中, 首先使用 evaluationScheme() 函数将数据分为 3 个子集。然后, 将要比较的多个模型对应的参数用一个列表 algorithms2compare 表示, 该列表的每个元素也是一个列表, 对应一个模型包含的参数, 其中 name 是算法的类型, param 是该算法对应的具体参数。接着直接调用 evaluate() 函数就可以按照 evaluationScheme() 中的数据划分来训练、比较不同的推荐模型。在使用 evaluate() 函数时, 可以指定 type 参数为 'topNList' 或 'ratings', 表示以何种方式调用 predict() 函数。当 type 参数为 'topNList' 时, 还可以指定参数 n, 表示将对诸模型返回的推荐结果的前 n 个分别计算评价指标并比较。在上面的代码中, 我们将 n 设为 $c(1, 3, 5, 10, 15, 20)$, 表示将对这 6 种不同的情况进行比较。最后, 可以调用 plot() 函数来直接可视化不同的模型的结果。图 7-9 显示了 5 种不同的算法在 n 等于 1、3、5、10、15、20 这 6 种情况下的 ROC 曲线 [见图 7-9 (a)], 以及召回率-精确率曲线 [见图 7-9 (b)]

的比较。图 7-10 则给出了在 Jester5k 数据上不同算法的 RMSE、MSE 和 MAE 比较。

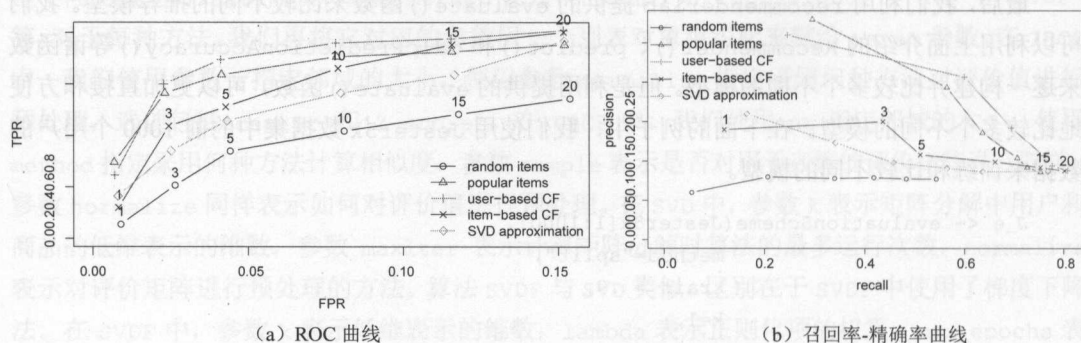


图 7-9 不同算法在 Jester5k 数据集上的比较

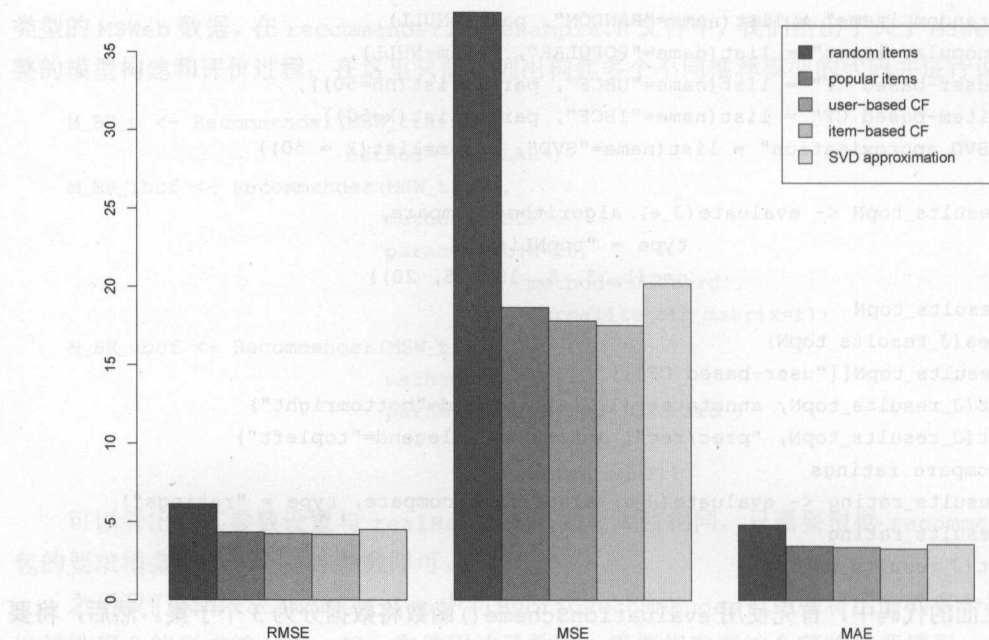


图 7-10 在 Jester5k 数据上不同算法的 RMSE、MSE 和 MAE 比较

7.6 推荐算法的评价和选取

在本节中，我们回顾一下上面介绍的各种推荐算法，并讨论在实际中如何选取合适的算法。这里我们着重讨论协同过滤的推荐算法。

在多次关于推荐问题的数据挖掘比赛中，如 Netflix Prize、KDD Cup 等，基于矩阵分解的推荐算法都取得了良好的表现。对于很多数据，基于矩阵分解的算法都能取得较好的效果，

值得一试。但是我们也要指出, 基于矩阵分解的算法并不一定适用于所有的数据。在 Netflix Prize 中, 采用的是 RMSE 作为算法的评价标准。如果采用不同的标准, 那么基于矩阵分解的算法未必是最优的算法。此外, 基于矩阵分解的算法对于参数比较敏感。在实际使用基于矩阵分解的算法时, 我们要根据数据的具体特征仔细地调节参数。很多时候, 在部署推荐算法时, 随着时间的变化, 数据一般也会发生变化。因此, 在实际部署的过程中, 我们要根据数据的变化及时调整参数, 从而取得较好的性能。

与基于矩阵分解的推荐算法相比, 基于邻域的推荐算法简单直观, 易于实现。特别是基于商品的推荐算法, 对于推荐结果的解释也很容易。在采用基于邻域的推荐算法时, 一个非常实际问题是: 是采用基于用户的邻域推荐算法还是采用基于商品的邻域推荐算法?

其中一个影响算法选择的因素是推荐结果的准确性。很多时候, 我们需要考虑用户数目和商品数目的对比。在基于用户的方法中, 我们依赖于用户-用户之间的相似度; 而在基于商品的方法中, 我们依赖于商品-商品之间的相似度。通过前面关于相似度显著性的讨论我们可以知道, 相似度的计算存在着“可靠性”的问题。在基于邻域的推荐系统中, 少量“可靠”的相似用户或者商品的作用远大于大量“不可靠”的用户或者商品的作用。因此, 如果用户数目远大于商品数目, 一般而言, 商品-商品之间的相似度度量会更可靠一些。此外, 基于商品的推荐算法在商品-商品相似度的基础上, 利用每个用户以往对其他商品的评价情况来推断该用户对某一商品的喜好程度。简而言之, 就是利用用户自己的信息来推断自己的喜好。而基于用户的推荐算法则是利用其他用户对于某一商品的喜好程度来推断该用户对于这一商品的喜好程度, 而其他用户的评价习惯等可能与该用户不完全一样, 导致在推断时可能存在较大的误差, 从而得到较差的推荐结果。因此, 在很多实际问题中, 基于商品的推荐算法更容易得到更好的推荐结果。当然, 基于商品的推荐算法并不是绝对好于基于用户的推荐算法。在实际中, 很多情况下基于用户的算法要优于基于商品的算法。我们需要根据实际问题的需要和数据的具体特征选择相应的算法。

另一个不同点是基于商品的方法通常给出比较稳妥但是没有太多新意的推荐, 而基于用户的方法则会给出一些新颖甚至不是那么“安全”的推荐。

在基于商品的方法中, 我们估计用户 u 对商品 i 的喜好程度 r_{ui} 时, 首先考虑用户 u 评价过的所有商品, 从这些商品中选出与商品 i 最相似的 k 件商品, 根据用户对这些商品的喜好来估计 r_{ui} 。因此, 在推荐新商品时, 推荐系统更倾向于推荐那些与用户已经熟悉的商品相似的商品。例如, 在 Netflix Prize 的电影推荐中, 使用基于商品的邻域方法更倾向于推荐与用户看过的电影相似的电影, 如相同的类型、包含同样的演员等。这样的推荐一般来说都是合理的, 但是很难向用户推荐比较“新”的商品。

而在基于用户的邻域方法中, 则很有可能推荐一些更加“新颖”的商品。这里举一个简单的例子, 假设一个用户 u 喜欢不同类型的电影, 而用户 v 目前只观看过一种类型的电影, 如果用户 u 和用户 v 很相似的话, 使用基于用户的方法, 用户 u 喜欢的电影可以推荐给用户 v 。

我们还可以从“可解释性”角度来进一步对比这两种方法。基于商品的方法一般给出的推荐都是比较安全的, 并且很好解释。对于用户来说, 基于商品的方法所使用的数据主要是

基于自己的过往数据,非常直观。例如,在电影推荐中,对用户 u 推荐典型电影 $M1$ 可以很容易解释:因为用户 u 之前看过与 $M1$ 相似度很高的若干电影 $M2$ 、 $M3$ 等。而基于用户的方法,则不是那么直观,因为每个用户并不了解其他用户的历史数据。

还有一个因素是计算复杂度。一般来讲,在很多实际问题中,用户的数目远大于商品的数目。由于基于商品的推荐算法只需要计算商品-商品的相似度,因此需要的计算资源会较少。衡量计算复杂度的另一个重要因素是系统的更新速度。注意,当我们计算同时评价过同一商品的两个用户之间的相似度时,他们同时评价过的商品数目一般而言不是很多。而我们计算两件商品之间的相似度时,同时评价过这两件商品的用户数目一般要大得多。因此,当新增一些评价值时,用户之间的相似度值可能会发生很大变化,而商品之间的相似度则基本上不受影响。此外,在很多实际的推荐系统中,用户的增加是很频繁的,而商品的增加则不是那么频繁。因此,用户-用户相似度要比商品-商品相似度的更新频率高得多。在这种情况下,基于商品的推荐算法相应的计算复杂度要低很多。

这里我们还特别指出评价值中商品的长尾分布对基于用户的邻域推荐算法的影响。在计算用户之间的相似度时,主要依赖于评价信息。这样的话,那些流行的商品的评价信息就主导了用户相似度的计算。换言之,我们基本上是利用那些流行的商品来确定用户之间是否相似。在实际中,虽然用户 u 和用户 v 对于那些流行商品的喜好一致,但是很有可能他们对于那些不流行的商品的喜好完全不同。

总之,与前面讨论的分类和回归算法类似,合理的算法的选择依赖于数据的具体特征。因此,在实际使用中,读者需要根据实际需要灵活选择算法。

第 8 章 排序学习

8.1 排序学习简介

在本章中，我们主要讨论排序学习（learning to rank）。排序学习在很多领域有着广泛的应用，如信息检索（information retrieval, IR）、自然语言处理（natural language processing, NLP）。最常见的例子是文档检索（document retrieval）。本章主要从机器学习的角度讨论如何解决排序问题。

我们以文档检索为例说明什么是排序学习。在文档检索中，我们有一个文档集合 $\{d_1, d_2, \dots, d_N\}$ 。用户提供一个查询 q （一般由若干词组成），然后从文档集合中返回最相关的文档。根据查询 q 中包含的词，找出那些含有这些词的文档，并将这些文档按照与查询的相似程度排好序，最后选择相似度最高的若干文档返回，即作为查询结果。图 8-1 描述了文档检索的过程。

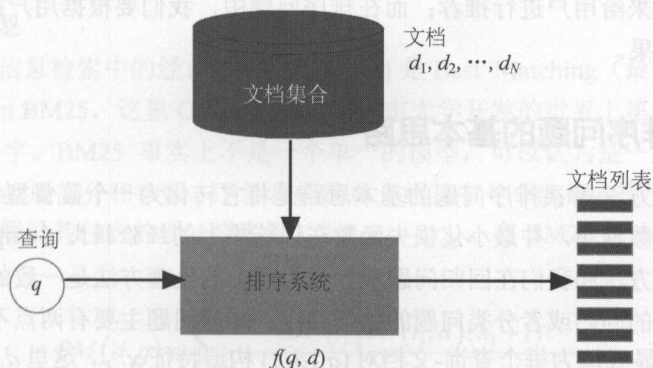


图 8-1 文档检索

在这个文档排序的例子中，需要估计排序模型 $f(q, d)$ ，这里 q 是查询， d 是文档， f 是我们学习得到的排序模型。在很多传统的信息检索模型中， f 都是通过一些经验公式而不是学习得到的。例如，在最简单的情况下，可以将 q 也视为一个较小的文档，利用词频-逆文档频率（TF-IDF）为查询和文档构建相应的特征，并将查询 q 和文档 d 之间的相似度作为 $f(q, d)$ 。使用经验公式的缺点是很难根据数据和问题的具体特性采用针对性的方法以提高性能。

通过使用机器学习，可以从数据中学习出模型 $f(q, d)$ 。现在，基于机器学习的模型在排序问题中越来越流行，其原因如下：

- (1) 使用机器学习方法可以考虑多种不同的信号。例如，在网络搜索中，可以将 PageRank

算法和基于文档的排序方法很好地结合起来。而在传统的信息检索方法中,很难同时引入多种数据。

(2) 随着数据越来越大,很多常用的信息检索方法难以适用。而机器学习的方法能够从海量数据中自主学习出模型,因此受到了广泛的欢迎。而且在实际问题中,数据经常发生变化,在这种情况下,我们经常需要根据数据调整模型。在这种情况下,机器学习方法具有显著的优势。

与前面讨论的分类问题和回归问题类似,我们也将排序学习转化为监督型学习(supervised learning)的任务。因此,在排序学习中,我们也有训练数据和测试数据。但是与分类和回归问题不同,排序学习的训练集中包括查询和文档。每个查询和若干个文档相关联。在训练集中,文档和查询的相关度也给出。这里的相关度可以用多种方式来表示。例如,可以是相关或者不相关,也可以是相关的程度,如从1到5的值。

与前面的分类或者回归问题不同,排序学习中的一个重要的概念是查询。用户首先提供查询信息,然后系统根据用户的查询,从对象集合中返回与用户查询最匹配的若干对象。例如,在使用搜索引擎时,对象集合就是整个互联网上的网页的集合。当用户在 expedia.com 上搜索航班时,这里的对象集合就是所有航班的集合;当用户在 expedia.com 上搜索旅馆时,这里的对象集合就是所有旅馆的集合。

查询也是排序问题和推荐问题的区别之一:在推荐问题中,用户没有查询,我们通过考察用户的历史信息来给用户进行推荐;而在排序问题中,我们要根据用户的查询,找出最匹配其输入查询的结果。

8.1.1 解决排序问题的基本思路

使用机器学习方法解决排序问题的基本思路是将它转化为一个监督型学习问题。具体地讲,就是考虑训练数据集,并最小化损失函数在训练集上的经验损失(empirical loss)。这种最小化损失函数的方法和我们在回归问题和分类问题上的处理方法是一致的。

但是,与前面的回归或者分类问题的处理相比,排序问题主要有两点不同。

(1) 我们需要显式地为每个查询-文档对 $(q_i, d_{i,j})$ 构造特征 $\mathbf{x}_{i,j}$,这里 $d_{i,j}$ 是一个与查询 q_i 对应的文档。

(2) 在排序问题中,由于算法评价标准不同,因此不能直接使用回归或者分类问题中使用的损失函数。

在排序问题中采用的算法评价指标通常都是不连续和不可导的。因此,这些指标对应的损失函数也是不连续和不可导的,这使得很多数值优化算法无法直接应用。在这种情况下,通常采用一些近似函数来逼近那些算法评价指标。进一步,按照排序算法的具体实现方式,可以分为如下3类。

- 逐点(pointwise)方法:在逐点方法中,每次只考虑一个查询和一个文档,以及对应的评分或者相关度;

- 逐对 (pairwise) 方法: 在逐对方法中, 每次考虑同一查询对应的一对文档和它们对应的相关度的差;
- 逐列 (listwise) 方法: 在逐列方法中, 每次考虑一个查询对应的一个文档序列和它们对应的排序结果。

基本上, 逐点方法是较早提出的适用于排序问题的算法, 而且很多算法都是从已有的分类和回归算法借鉴过来的。而逐对和逐列方法是后期发展出来的专门针对排序问题的算法。

8.1.2 构造特征

在使用机器学习算法解决排序问题时, 核心问题是研究查询和文档之间的关系。因此, 需要给查询-文档对构建相应的特征。具体来说, 我们要为每个查询-文档对 $(q_i, d_{i,j})$ 构造特征 $\mathbf{x}_{i,j}$, 这里 $\mathbf{x}_{i,j}$ 是一个向量:

$$\mathbf{x}_{i,j} = \phi(q_i, d_{i,j}) \quad (8-1)$$

这里用 ϕ 表示构造的函数。

在实际中, 我们有多种方法可以构造特征。例如, 可以使用信息检索中传统的 BM25 模型的输出作为一个特征, 也可以使用 PageRank 对于网站的权重作为一个特征。这里我们介绍 BM25 模型的输出和一些在文档检索中常用的特征。

1. BM25 模型

BM25 模型是信息检索中的经典模型, 这里 BM 是 Best Matching (最优匹配) 的简写。BM25 也称为 Okapi BM25, 这里 Okapi 是由伦敦城市大学开发的世界上第一个使用该模型的信息检索系统的名字。BM25 事实上不是一个单一的模型, 可以认为是一组排序模型; 而模型之间具有不同的组成部分和参数。下面我们简单介绍其中一个流行的模型。

对于查询 q , 假设其包含的项 (或者词) 为 t_1, t_2, \dots, t_M 。在 BM25 算法中, 我们计算 q 和文档 d 的相似度为:

$$BM(d, q) = \sum_{i=1}^M \frac{IDF(t_i) TF(t_i, d) (k_1 + 1)}{TF(t_i, d) + k_1 \left(1 - b + b \frac{LEN(d)}{avdl} \right)} \quad (8-2)$$

这里 $TF(t_i, d)$ 是项 t_i 出现在文档 d 中的频率, $LEN(d)$ 是文档 d 的长度 (可用文档所包含的词的总数度量), $avdl$ 是文档集中文档的平均长度, k_1 和 b 是控制参数, $IDF(t_i)$ 是项 t_i 对应的逆文档频率 (Inverse Document Frequency, IDF) 权重:

$$IDF(t_i) = \ln \frac{N}{n(t_i)} \quad (8-3)$$

这里 N 是所有文档的总数, $n(t_i)$ 是其中包含项 t_i 的文档数目。在信息检索中, 对于 $TF(t_i, d)$ 和 $IDF(t_i)$ 都有一些不同的定义, 我们这里介绍的都是最简单的定义。对于控制参数 k_1 和 b , 一

一般而言, 需要根据数据具体调节。一般的推荐值为 $k_1 \in [1.2, 2.0]$, $b = 0.75$ 。

2. 其他特征

表 8-1 总结了一些适用于文档检索的常用特征, 其中 $w \in q \cap d$ 表示查询 q 和文档 d 相交的词 w , $TF(w, d)$ 表示 w 在 d 中出现的词频, $IDF(w)$ 表示 w 的逆文档频率, $LEN(d)$ 表示文档 d 的长度。对于其他问题, 我们可以类似地建立相应的特征。

表 8-1 适用于文档检索的常用特征

特征编号	特征定义
1	$\sum_{w \in q \cap d} TF(w, d)$
2	$\sum_{w \in q \cap d} \ln(TF(w, d) + 1)$
3	$\sum_{w \in q \cap d} \frac{TF(w, d)}{LEN(d)}$
4	$\sum_{w \in q \cap d} \ln\left(\frac{TF(w, d)}{LEN(d)} + 1\right)$
5	$\sum_{w \in q \cap d} \ln(IDF(w))$
6	$\sum_{w \in q \cap d} \ln(\ln(IDF(w)))$
7	$\sum_{w \in q \cap d} \ln\left(\frac{TF(w, d)}{LEN(d)} \ln(IDF(w)) + 1\right)$
8	$\sum_{w \in q \cap d} TF(w, d) \ln(IDF(w))$

8.1.3 获取相关度分数

注意, 在训练集中, 对于查询-文档对 $(q_i, d_{i,j})$, 我们已知相关度 $y_{i,j}$ 。在实际中, 通常有两种方式获得。第一种是人工标注的方法。这种方法适用于数据规模较小的情况。当数据规模较大时难以实行。第二种方式是间接方法。例如, 我们可以通过搜索的日志文件来得到。例如, 在网站 expedia.com 上, 用户搜索某地的旅馆信息后, expedia.com 返回一系列结果。在实际中, 用户会点击一部分旅馆, 显示对这些旅馆感兴趣。如果用户最终预定了某一旅馆, 则表示用户对该旅馆的各方面包括位置、价格等都比较满意。另外, 如果某些旅馆虽然排在返回结果的前几名, 但是用户都没有点击, 则反映了用户对这些旅馆没有兴趣。在这个例子中, 我们可以得到如下的排序结果:

预定的旅馆 > 点击但没有预定的旅馆 > 没有点击的旅馆

在本章中, 我们主要使用文档检索来讨论算法。一方面, 文档检索是排序问题中的一个常见问题; 另一方面, 文档检索中场景比较简单, 利于我们讨论算法的原理。接下来, 我们首先介绍本章使用的数学符号, 然后在后续章节中, 介绍排序算法的评价指标, 并逐一介绍

逐点方法、逐对方法和逐列方法。

8.1.4 数学符号

查询一般用 q 表示, 并将查询的集合记为 Q 。我们用 d 表示文档, 用 D 表示文档的集合。在训练集中的查询集合记为 $\{q_1, q_2, \dots, q_m\}$ 。对于查询 q_i , 我们将训练集中与 q_i 对应的文档集合记为 $D_i = \{d_{i,1}, d_{i,2}, \dots, d_{i,N_i}\}$ 。注意, 对于查询 q_i , 我们有 N_i 个文档与之对应, 其中有些文档与 q_i 相关, 有些则不相关。在文档集合 D_i 中, 我们将所有与 q_i 相关的文档的总数记为 n_i 。

对于训练集中的查询-文档对 $\{q_i, d_{i,j}\}$, 其对应的相关度为 $y_{i,j}$, 我们把为该查询-文档对构建的特征记为 $\mathbf{x}_{i,j}$ 。相关度 $y_{i,j}$ 可以为布尔型, 也可以使用多个不同的分数表示相关程度。在本章中, 我们假设这个分数越高, 查询和文档的相关度越高。我们把查询 q_i 对应的所有文档的相关度分数记为向量的形式 $\mathbf{y}_i = [y_{i,1}, y_{i,2}, \dots, y_{i,N_i}]^T \in \mathbb{R}^{N_i}$ 。举例来说, 假设对于查询 q_1 有 3 个对应文档 $d_{1,1}, d_{1,2}, d_{1,3}$, 其中只有第二个文档 $d_{1,2}$ 与 q_1 相关, 则对应的相关度向量可设置为 $\mathbf{y}_1 = [0, 1, 0]^T$ 。

在逐对方法中, 我们要考察每个查询 q_i 对应的所有文档对。我们将查询 q_i 所对应的所有文档对的集合记为 P_i 。对于训练集中的任意文档对 $(d_{i,u}, d_{i,v})$, 我们假设第一项 $d_{i,u}$ 比第二项 $d_{i,v}$ 更相关, 记为 $d_{i,u} \triangleright d_{i,v}$ 。在前一个例子中, q_1 有 3 个对应文档 $d_{1,1}, d_{1,2}, d_{1,3}$, 其中只有第二个文档 $d_{1,2}$ 与 q_1 相关, 则对应的文档对集合为 $P_i = \{d_{1,2} \triangleright d_{1,1}, d_{1,2} \triangleright d_{1,3}\}$ 。

我们把排序模型记为 f 。一般而言, 我们利用特征 $\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \dots, \mathbf{x}_{i,N_i}$ 作为输入。 f 的输出可以有多种, 其中最常用的是 $f(\mathbf{x}_{i,j})$ 是查询 q_i 与文档 $d_{i,j}$ 的相关度。根据排序模型 f , 我们可以将与 q_i 对应的所有文档 $d_{i,1}, d_{i,2}, \dots, d_{i,N_i}$ 按照输出的相关度从高到低排列好输出。如果我们将文档集合 D_i 中的文档用整数 $1, 2, \dots, N_i$ 来标记, 那么利用模型 f 就会得到一个关于 $1, 2, \dots, N_i$ 的排列。我们将 $1, 2, \dots, N_i$ 的所有排列的集合记为 Π_i 。对于某一系列 $\pi \in \Pi_i$, 我们记 $\pi(r)$ 为排列 π 中的第 r 个文档。例如, 对于排列 $1, 2, \dots, N_i$, 我们有 $\pi(r) = r$, 这里 $r = 1, 2, \dots, N_i$ 。对于排列 $N_i, N_i-1, \dots, 1$, 我们有 $\pi(r) = N_i + 1 - r$, 这里 $r = 1, 2, \dots, N_i$ 。对于查询 q_i , 不失一般性, 我们将其所对应的所有文档的真实排序记为 π_i 。

8.2 排序算法的评价

鉴于排序问题的特殊性, 我们不能直接使用分类或者回归问题中的算法评价指标。在本节, 我们介绍常用的评价排序算法的度量或者指标, 包括 MAP (mean average precision)、DCG (discounted cumulative gain) 和 NDCG (normalized discounted cumulative gain)。基本原理是: 首先对于每个查询, 我们比较排序模型返回的结果和真实结果, 并计算相应的度量; 在将这些度量对所有的查询平均后, 我们得到最后的评价结果。下面依次详细介绍 MAP、DCG 和 NDCG。

8.2.1 MAP

MAP 是信息检索中常用的一个评价排序模型的度量。在 MAP 中, 我们假设文档与查询的相关度为 0 或者 1。计算 MAP 时, 我们首先计算测试集中每个查询对应的平均精度 (average precision, AP), 然后计算所有查询的 AP 的平均值作为最终的 MAP。

首先, 计算对于查询 q_i 所得的排序结果中前 k 个文档的精度 (precision at k , $P@k$):

$$P@k(q_i) = \frac{n_i(k)}{k} \quad (8-4)$$

这里 $n_i(k)$ 是模型返回结果前 k 个文档中与查询 q_i 真正相关的文档数目。

在 $P@k(q_i)$ 的基础上, 可以计算查询 q_i 所对应的平均精度 (average precision, AP), 其定义为:

$$AP(q_i) = \frac{\sum_{k=1}^{N_i} P@k(q_i) \cdot l_k}{n_i} \quad (8-5)$$

这里 N_i 是对于查询 q_i 我们考察的文档总数 (如排序算法为查询 q_i 所返回的文档总数), n_i 是其中与 q_i 相关联的文档总数。 l_k 表示模型所得到的排序结果中第 k 位的文档是否与查询 q_i 相关。如果相关, $l_k = 1$, 否则 $l_k = 0$ 。换言之, 我们只考虑返回结果中那些真正相关的文档所对应位置的 $P@k(q_i)$ 。

最后, 在为每个查询 q_i 计算 $AP(q_i)$ 后, 我们计算所有 $AP(q_i)$ 的平均值, 得到 MAP:

$$MAP = \frac{1}{m} \sum_{i=1}^m AP(q_i) \quad (8-6)$$

下面我们使用一个实际例子来说明如何计算 MAP。

例 8-1 在这个例子中, 假设我们有一个查询, 其对应 8 个文档, 编号为 1~8。我们假设有两个排序模型 M1 和 M2, 其排序结果分别为 87654321 和 12345678。根据表 8-2, 其中与查询相关的文档为 1、6 和 7。我们分别计算这两个模型对应的 MAP。为了简化讨论, 在这个例子中我们假设只有一个查询, 因此只需要计算该查询对应的 AP。

表 8-2 计算 MAP 示例

文档编号	真实相关度	排序模型 M1	排序模型 M2
1	1	8	1
2	0	7	2
3	0	6	3
4	0	5	4
5	0	4	5
6	1	3	6
7	1	2	7
8	0	1	8

表 8-3 展示了模型 M1 对应的 AP 的计算过程。在表 8-3 中, 对于每个 k , 首先计算对应的 $P@k$, 然后计算对应的 $P@k \times l_k$, 最后将所有的 $P@k \times l_k$ 累加得到:

$$\sum_{k=1}^{N_i} P@k(q_i) \cdot l_k = 0.5 + 0.667 + 0.375 = 1.542$$

注意, 一共有 3 个文档相关, 因此 $n_i = 3$, 则对应的 AP 为:

$$AP = \frac{1.542}{3} = 0.514$$

表 8-3 模型 M1 对应 AP 的计算过程

k	文档编号	l_k	M1 排序	$P@k$	$P@k \times l_k$
1	8	0	1	0	0
2	7	1	2	0.5	0.5
3	6	1	3	0.667	0.667
4	5	0	4	0.5	0
5	4	0	5	0.4	0
6	3	0	6	0.333	0
7	2	0	7	0.286	0
8	1	1	8	0.375	0.375

相应地, 表 8-4 展示了模型 M2 对应的 MAP 的计算过程。将所有的 $P@k \times l_k$ 累加得到:

$$\sum_{k=1}^{N_i} P@k(q_i) \cdot l_k = 1 + 0.333 + 0.429 = 1.762$$

注意 $n_i = 3$, 因此对应的 AP 为:

$$AP = \frac{1.762}{3} = 0.587$$

表 8-4 模型 M2 对应 AP 的计算过程

k	文档编号	l_k	M2 排序	$P@k$	$P@k \times l_k$
1	1	1	1	1	1
2	2	0	2	0.5	0
3	3	0	3	0.333	0
4	4	0	4	0.25	0
5	5	0	5	0.2	0
6	6	1	6	0.333	0.333
7	7	1	7	0.429	0.429
8	8	0	8	0.375	0

从模型 M1 和 M2 的对比可以看出, 虽然 M1 把第 2、3 位的文档都排对了, 但是 M2 把最重要的第 1 位的文档选对了, 因此 M2 对应的 MAP 更大一些。

8.2.2 DCG

在前面的 MAP 中, 我们只考虑了返回的文档是否与查询相关。在实际问题中, 我们有时需要考虑不同的相关度。在前面那个关于 expedia.com 的例子中, 我们就考虑了多种情况。

(1) 用户点击了返回的旅馆链接并预定了旅馆;

(2) 用户仅点击但是并没有预定旅馆;

(3) 用户没有点击旅馆链接并且没有预定旅馆。

在这个例子中, 对于不同的情况, 我们应该给予不同的权值。例如, 对于情况 1, 我们可以给予 5 分, 对于情况 2, 我们可以给 1 分, 对于情况 3, 我们可以给 0 分。此外, 我们还要根据上述几种情况在返回结果中的位置给予不同的权重。DCG 综合考虑了返回的序列中每个文档与查询的相关度, 并对位置信息直接予以考虑。

与 MAP 类似, 在计算 DCG 时, 首先计算每个查询对应的 DCG, 然后对所有查询求平均。假设对于查询 q_i 模型返回的排列结果是 π_i , 那么位置 k 的 DCG 定义为:

$$DCG@k(q_i) = \sum_{j=1}^k G(\pi_i^{-1}(j)) \eta(j) \quad (8-7)$$

这里我们顺次考虑排序结果 π_i 中的第 j ($j=1, 2, \dots, k$) 个文档, $\pi_i^{-1}(j)$ 表示在排列 π_i 中排在第 j 位的文档, $G(\pi_i^{-1}(j))$ 表示该文档对应的权值, 而 $\eta(j)$ 表示由位置 j 决定的权值。对于 $G(\pi_i^{-1}(j))$, 我们可以根据实际问题的性质给予不同的权值。一种常用的关于 $G(\pi_i^{-1}(j))$ 的计算公式为:

$$G(\pi_i^{-1}(j)) = 2^{l_{\pi_i^{-1}(j)} - 1} \quad (8-8)$$

这里 $l_{\pi_i^{-1}(j)}$ 表示排列 π_i 中排在第 j 位的文档与查询 q_i 的相关度。例如, 如果相关, 则 $l_{\pi_i^{-1}(j)} = 1$, 否则为 0。此外, $l_{\pi_i^{-1}(j)}$ 也可以取其他值, 例如, 在实际问题中可以根据文档与查询不同的相关度, 设定取值范围为 $\{3, 2, 1, 0\}$ 。在前面的关于 expedia.com 的例子中, 取值范围为 $\{5, 1, 0\}$ 。

下面是一个常见的 $\eta(j)$ 计算公式:

$$\eta(j) = \frac{1}{\log_2(j+1)} \quad (8-9)$$

例 8-2 在表 8-5 所示的这个例子中, 我们使用前面一个例子中的 M1 模型的输出来计算 $DCG@k$ 。这里我们假设 $l_{\pi_i^{-1}(j)} = 1$ 或者 0, $\eta(j)$ 的计算使用式 (8-9)。

表 8-5 模型 M1 对应 DCG 的计算过程

k	文档编号	$l_{\pi_i^{-1}(j)}$	M1 排序	$G(\pi_i^{-1}(j))$	$\eta(j)$	$DCG@k$
1	8	0	1	0	1.000	0
2	7	1	2	1	0.6309	0.6309
3	6	1	3	1	0.5	1.1309
4	5	0	4	0	0.4307	1.1309
5	4	0	5	0	0.3869	1.1309
6	3	0	6	0	0.3562	1.1309
7	2	0	7	0	0.3333	1.1309
8	1	1	8	1	0.3155	1.4464

在这个例子中，如果我们考虑 $k=8$ ，则有 $DCG@8=1.4464$ 。在我们提供的 R 程序 `ranking_metrics.R` 中，也提供了计算 DCG 的函数。

8.2.3 NDCG

在 DCG 的基础上，可以将其标准化到 0 和 1 之间，成为 NDCG。NDCG 是目前在排序学习中应用非常广泛的一种度量。与 MAP 和 DCG 类似，我们也是先计算每个查询 q_i 对应的 NDCG，然后对所有的查询求平均值得到最后的结果。在计算关于查询 q_i 的 $NDCG@k$ 时，核心是先计算 $DCG@k$ ，然后寻找使得 $DCG@k$ 最大的排列，计算相应的 $DCG@k$ ，记为 $z_{max}(k)$ ，那么 NDCG 的定义为：

$$NDCG@k(q_i) = \frac{DCG@k(q_i)}{z_{max}(k)} \quad (8-10)$$

根据 NDCG 的定义，我们知道其值在 0 和 1 之间。

例 8-3 在下面的这个例子中，我们仍旧使用前面一个例子中的 M1 模型的输出来计算 $DCG@k$ 。我们只需要计算最好的排序结果对应的 $DCG@k$ 即可。最好的排序结果对应的相关度序列应该为 1,1,1,0,0,0,0。其对应的 $DCG@8=2.1309$ 。因此，用 M1 模型的输出来计算 $NDCG@8$ 的结果如下：

$$NDCG@8 = \frac{1.4464}{2.1309} = 0.6788$$

本书附属的 R 程序 `ranking_metrics.R` 中也提供了直接计算 NDCG 的函数。

8.2.4 讨论

从上面讨论的若干排序算法评价度量来看，它们基本上满足如下性质。

(1) 基本上所有的指标都是先对每个查询计算, 然后对测试集中所有的查询进行平均, 进而得到最终结果。因此, 每个查询都被平等对待。

(2) 所有这些指标都是基于位置的。换言之, 就是在计算度量时考虑了返回结果中文档在不同位置的权重。

注意, 在排序学习中, 所有算法评价指标都是基于位置的。但是从数学上来讲, 这些评价指标都是很难直接优化的。这里我们考虑如下情况: 假设我们的排序模型 f 返回了两个文档 d_1 、 d_2 及其对应的分数 $s_1 = f(\mathbf{x}_1)$ 、 $s_2 = f(\mathbf{x}_2)$, 这里 $s_1 > s_2$, \mathbf{x}_1 、 \mathbf{x}_2 分别是文档 d_1 、 d_2 对应的特征。我们将 s_2 逐渐增大, 但是在 s_2 大于 s_1 之前, 排序模型的返回结果是一样的。也就是说, 在 s_2 增大到 s_1 的过程中, 算法的性能指标没有任何变化。但是一旦 $s_2 > s_1$, 算法的性能指标会发生很大变化。这意味着如果我们直接最小化这些度量的话, 这些度量都是不连续且不可导的。而我们在机器学习中常用的策略是通过计算导数来最小化目标函数从而求出最优解。因此, 在排序学习中, 我们需要新的办法来最优化算法的性能。在下面的章节中, 我们将分别介绍逐点方法、逐对方法和逐列方法, 并重点讨论较为常用的逐对方法以及在 R 中的实际使用。

8.3 逐点方法

逐点方法是解决排序问题最简单直接的方法。在逐点方法中, 我们将排序问题转化为回归或者分类问题。这样前面讨论的那些适用于回归或者分类的算法就可以直接使用了。按照转化问题的类型, 逐点方法可以进一步分为以下 3 种。

(1) 基于回归的方法;

(2) 基于分类的方法;

(3) 基于有序回归的方法。

在基于回归的方法中, 我们直接将每个查询-文档对 $(q_i, d_{i,j})$ 对应的相关度 $y_{i,j}$ 作为实数型的输出来构建模型。在基于分类的方法中, 我们将 $y_{i,j}$ 的每个不同值视为一个不同的类标(class label), 从而构建一个分类模型。注意, 在此方法中, 我们并不考虑 $y_{i,j}$ 之间的顺序关系。在基于有序回归的方法中, 我们将 $y_{i,j}$ 看成不同的类, 但是类之间存在顺序关系。

将前面讨论的回归或者分类算法应用于排序问题时应注意以下两点。

(1) 输入数据不同。我们需要显式地为每个查询-文档对 $(q_i, d_{i,j})$ 构造特征 $\mathbf{x}_{i,j}$ 。

(2) 在排序问题中, 为了得到最优性能, 一般不能直接使用回归或者分类问题中使用的损失函数。我们需要更加适用于排序问题的损失函数。

在本节中, 我们介绍两种基于 SVM 的逐点排序算法以帮助读者理解逐点方法的基本原理。在此基础上, 我们进一步讨论逐点排序算法的优缺点。

在下面的讨论中, 假设我们有 m 个查询 $\{q_1, q_2, \dots, q_m\}$ 。对于查询 q_i , 我们将训练集中与 q_i 对应的文档集合记为 $D_i = \{d_{i,1}, d_{i,2}, \dots, d_{i,N_i}\}$, 其相应的相关度记为 $y_i = \{y_{i,1}, y_{i,2}, \dots, y_{i,N_i}\}$ 。将

查询-文档对 $(q_i, d_{i,j})$ 构建的特征记为 $\mathbf{x}_{i,j}$ 。

8.3.1 基于 SVM 的逐点排序方法

在将 SVM 推广到逐点排序方法的过程中,有两种思路:第一种思路称为定间隔策略(fixed margin strategy),第二种策略称为“间隔和”策略(sum of margins strategy)。下面我们分别予以讨论。

在讨论新的算法之前,我们简单回顾一下标准的 SVM 算法。在标准的 SVM 中,我们要尽量使得每个样本被正确地分类。回忆一下在基本的 SVM 中,对于第 i 个样本 \mathbf{x}_i 以及对应的类标 $y_i (y_i = \pm 1)$, 我们构建了如下的约束条件:

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \varepsilon_i, \quad \varepsilon_i \geq 0 \quad (8-11)$$

这里我们使用松弛变量 $\varepsilon_i \geq 0$ 来表示样本 \mathbf{x}_i 是否被正确分类以及被错误分类时错误的程度。注意,在这里我们稍稍“滥用”了一下符号,使用了 \mathbf{x}_i 、 y_i 、 ε_i 、 b 等。

在排序问题中,我们也可以采取类似的思路,为排序问题构建类似的约束条件。在 Shashua 和 Levin 提出的适用于排序问题的 SVM 算法^[29]中,需要求解如下的优化问题:

$$\begin{aligned} \min & \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^m \sum_{j=1}^{N_i} \sum_{k=1}^{r-1} \left(\varepsilon_{j,k}^{(i)} + \varepsilon_{j,k+1}^{(i)*} \right) \\ \text{s.t.} & \quad \mathbf{w}^T \mathbf{x}_{i,j} + b_k \leq -1 + \varepsilon_{j,k}^{(i)}, \quad \text{如果 } y_{i,j} = k \\ & \quad \mathbf{w}^T \mathbf{x}_{i,j} + b_k \geq 1 - \varepsilon_{j,k+1}^{(i)*}, \quad \text{如果 } y_{i,j} = k+1 \\ & \quad \varepsilon_{j,k}^{(i)} \geq 0, \varepsilon_{j,k+1}^{(i)*} \geq 0, \quad i = 1, 2, \dots, m, \quad j = 1, 2, \dots, N_i, \quad k = 1, 2, \dots, r-1 \end{aligned} \quad (8-12)$$

这里我们假设相关度 $y_{i,j}$ 有 r 个不同的值 $1, 2, \dots, r-1, r$, 值越大, 相关度越高。与标准的 SVM 类似, 我们在这里要求解一个线性函数 $f(\mathbf{x}_{i,j}) = \mathbf{w}^T \mathbf{x}_{i,j} + b$ 。但是与标准 SVM 不同的是, 在标准 SVM 中我们只有两个类; 而在排序问题中我们有 r 个类标 $1, 2, \dots, r$, 且它们之间存在偏序关系。为了处理该偏序关系, 我们引入了 $r-1$ 个不同的截距 b_1, b_2, \dots, b_{r-1} , 对应 $r-1$ 个平行的超平面 $\mathbf{w}^T \mathbf{x} + b_k = 0, k = 1, 2, \dots, r-1$ 。在排序问题中, 我们将整个输入空间用 $r-1$ 个平行的超平面来划分, 使得类标为 k 的查询-文档对的特征 $\mathbf{x}_{i,j}$ 正好被映射到 $\mathbf{w}^T \mathbf{x}_{i,j} + b_{k-1} = 0$ 和 $\mathbf{w}^T \mathbf{x}_{i,j} + b_k = 0$ 之间。类似地, 我们引入松弛变量 $\varepsilon_{j,k}^{(i)}$ 和 $\varepsilon_{j,k+1}^{(i)*}$ 来处理不能被映射到指定区间所引起的惩罚。此外, 与标准的 SVM 类似, 我们也用 $\frac{1}{2} \|\mathbf{w}\|_2^2$ 来控制模型的复杂度。将这些综合起来, 我们使用 $\frac{1}{2} \|\mathbf{w}\|_2^2$ 和所有松弛变量的和作为最终的目标函数。

第二种策略称为“间隔和”策略。在这种策略中, 我们为每个类增加一个新变量 a_k , 这样使得类标为 k 的查询-文档对的特征 $\mathbf{x}_{i,j}$ 被映射到超平面 $\mathbf{w}^T \mathbf{x} + b_{k-1} = 0$ (下界) 和超平面

$w^T x + a_k = 0$ (上界) 之间, 如图 8-2 所示。

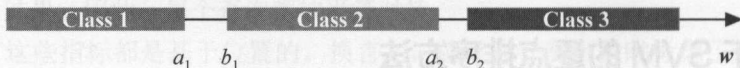


图 8-2 “间隔和”策略原理

我们希望 $x_{i,j}$ 在映射之后, 不同类对应的空间之间的间隔尽量大。例如, 在图 8-2 中, 我们希望 Class 1 和 Class 2 在投影之后的间隔 $b_1 - a_1$ 尽量大。一般来讲, 我们要求 $a_k - b_k$ 的和最小 (这就是该方法称为“间隔和”策略的原因)。另外, 我们也引入松弛变量以处理那些不能被正确分类的样本。因此, 在如下的优化问题中, 我们要最小化所有 $a_k - b_k$ 以及所有松弛变量的和:

$$\begin{aligned} \min & \sum_{k=1}^{r-1} (a_k - b_k) + C \sum_{i=1}^m \sum_{j=1}^{N_i} \sum_{k=1}^{r-2} (\varepsilon_{j,k}^{(i)} + \varepsilon_{j,k}^{(i)*}) \\ \text{s.t.} & \quad a_k \leq b_k \leq a_{k+1} \\ & \quad w^T x_{i,j} \leq a_k + \varepsilon_{j,k}^{(i)}, \text{ 如果 } y_{i,j} = k \\ & \quad w^T x_{i,j} \geq b_k - \varepsilon_{j,k}^{(i)*}, \text{ 如果 } y_{i,j} = k+1 \end{aligned} \quad (8-13)$$

$$\|w\|_2^2 \leq 1, \quad \varepsilon_{j,k}^{(i)} \geq 0, \quad \varepsilon_{j,k}^{(i)*} \geq 0, \quad i = 1, 2, \dots, m, \quad j = 1, 2, \dots, N_i, \quad k = 1, 2, \dots, r-1$$

8.3.2 逐点方法讨论

事实上, 逐点方法可以认为是分类或者回归问题在排序问题上的简单推广。当类别数 $r=2$ 时, 可以将逐点方法直接转化为分类问题来解决。另外, 逐点方法没有更多地挖掘排序问题本身的特性。例如, 在逐点方法中, 我们忽视了同一查询 q_i 所对应的多个文档 $d_{i,1}, d_{i,2}, \dots, d_{i,N_i}$ 之间的关系。

事实上, 在排序问题中, 我们更关注的是文档之间的相互位置而不是准确估计文档和查询之间的相关度。当然, 如果我们能够准确估计相关度, 也能得到较好的排序结果; 但是, 从解决问题的角度讲, 这要求我们首先解决一个更难的问题, 然后利用得到的答案构建排序结果。在实际中, 我们一般青睐更直接的方法。

在前面讨论的诸多评价排序算法性能的标准中, 基本上所有的标准都和查询 (query) 以及返回文档对应的排序位置 (position) 直接相关。这些标准都是先对每个查询计算, 然后对所有的查询计算平均值。在对每个查询计算时, 我们都考虑了返回的文档序列中每个文档的位置。因此, 在解决排序问题时, 有以下两个核心思想: (1) 直接考虑排序结果中每个文档的位置信息, 特别是前面几个位置; (2) 直接考虑查询, 包括每个查询所对应的文档以及它们之间的相互关系。

在逐点方法中, 我们没有在查询层次考虑模型优化。具体而言, 就是对于每个查询, 没有直接考虑和它相关的所有文档之间的相互关系。在逐点方法中, 我们将所有的查询和相应

的文档放在一起考虑。因此，如果有的查询对应很多的文档，则这样的查询就会在最终模型中造成很大的影响，而那些只对应较少文档的查询则被忽视。此外，我们没有直接考虑文档在返回结果中的位置。虽然我们使用了相关度分数来引入位置信息，但毕竟不是最直接的方式。

考虑到排序问题的特殊性，逐点方法并不是一种理想的方法。事实上，在实际中，逐点方法的使用也不多。因此，本节的主要目的是通过介绍两种 SVM 的变体来说明逐点方法的基本思想。

为了弥补逐点方法的不足，下面我们介绍逐对和逐列方法。在逐对方法中，我们显式地考虑了每个查询对应的文档两两之间的关系。在逐列方法中，我们显式地考虑了排序算法返回的整个排序序列。由于这两种方法直接考虑了每个查询以及对应文档之间的关系，因此能够获得更好的效果。

8.4 逐对方法

在逐对方法中，我们关注的是文档之间的相对排序：对于给定的文档 q_i ，如果我们有两个文档，到底哪个文档更加相关一些？在理想情况下，如果我们对于每个这样的问题都回答正确，就得到了一个完美的排序模型。

因此，我们可以很自然地将排序问题转化为分类问题：对于给定的查询 q_i 和对应文档对 $(d_{i,u}, d_{i,v})$ ，如果文档 $d_{i,u}$ 比 $d_{i,v}$ 更相关，则可以视为正类；否则视为负类。这样就可以利用分类算法来求解。但是要注意输入数据不是一个文档，而是一对文档。此外，与传统的分类算法类似，我们也可以在解决新的分类问题时选取不同的损失函数，如 Hinge 损失函数、指数损失函数、交叉熵损失函数等。

当我们将排序问题转化为输入数据是一对文档的分类问题时，很多算法都将最小化被错误分类的文档对的总数作为优化目标。但是，要注意最小化被错分的文档对的数目并不等于优化排序问题中的一些评价指标，如 NDCG。在本节中，我们介绍在处理逐对数据时直接优化排序算法评价指标的算法，如 LambdaRank 和 LambdaMART 算法。

在实践中，逐对算法是较常用和有效的排序算法，因此也是本章我们讨论的重点。此外，在实际中我们更容易得到适用于逐对方法的训练数据。事实上，对于真实的排序结果，很多时候我们并不知道一个完整的排序结果序列。在很多问题中训练数据以如下形式表示：对于查询 q_i ，我们知道文档 $d_{i,u}$ 应该排在文档 $d_{i,v}$ 之前。

本节首先介绍 SVM 的两种变形算法，并阐述逐对方法的一些基本原理，然后介绍基于人工神经网络的 RankNet 算法以及它的两种变形：LambdaRank 和 LambdaMART。其中 LambdaMART 算法在多次数据挖掘竞赛中都取得了非常优秀的成绩，因此在本节中我们将予以重点介绍。

8.4.1 Ranking SVM 算法

在 Ranking SVM 算法中，我们将标准的 SVM 算法推广到逐对方法中。假设对于查询 q_i ，

我们把它对应的任意文档对 $(d_{i,u}, d_{i,v})$ 所对应的特征对记为 $(\mathbf{x}_{i,u}, \mathbf{x}_{i,v})$ ，并把对应的类标记为 $y_{i,(u,v)}$ 。如果文档 $d_{i,u}$ 比文档 $d_{i,v}$ 与查询 q_i 更相关，即 $d_{i,u} \triangleright d_{i,v}$ ，则 $y_{i,(u,v)} = 1$ ；如果 $d_{i,v} \triangleright d_{i,u}$ ，则 $y_{i,(u,v)} = -1$ 。

在 Ranking SVM 中，假设模型表示为 $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ ，则我们要求解如下的优化问题：

$$\begin{aligned} \min & \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^m \sum_{u,v: y_{i,(u,v)}=1} \varepsilon_{i,(u,v)} \\ \text{s.t. } & \mathbf{w}^T (\mathbf{x}_{i,u} - \mathbf{x}_{i,v}) \geq 1 - \varepsilon_{i,(u,v)}, \text{ 如果 } y_{i,(u,v)} = 1 \\ & \varepsilon_{i,(u,v)} \geq 0, \quad i=1, 2, \dots, m, \quad d_{i,u} \triangleright d_{i,v} \in P_i \end{aligned} \quad (8-14)$$

与前面将标准 SVM 推广到逐点方法中类似，我们通过构建约束条件来为每对文档分类。对于查询 q_i 和相应的文档对所对应的特征向量 $(\mathbf{x}_{i,u}, \mathbf{x}_{i,v})$ ，如果 $d_{i,u}$ 比 $d_{i,v}$ 与 q_i 更相关，我们希望投影后 $\mathbf{w}^T \mathbf{x}_{i,u}$ 比 $\mathbf{w}^T \mathbf{x}_{i,v}$ 更大。类似地，我们也引入了松弛变量 $\varepsilon_{i,(u,v)}$ 。但要注意，这里我们使用的也是 Hinge 损失函数。根据前面的约束条件，假设 $y_{i,(u,v)} = 1$ ，即 $\mathbf{x}_{i,u}$ 比 $\mathbf{x}_{i,v}$ 与 q_i 更相关。如果 $\mathbf{w}^T \mathbf{x}_{i,u}$ 与 $\mathbf{w}^T \mathbf{x}_{i,v}$ 的差大于或等于 1，则惩罚项为 0；否则，计算 $1 - \mathbf{w}^T (\mathbf{x}_{i,u} - \mathbf{x}_{i,v})$ 作为惩罚项。最终的目标函数是所有松弛变量和 $\frac{1}{2} \|\mathbf{w}\|_2^2$ （该项对应着模型的复杂度）的和。注意，在约束条件中，我们只考虑了满足 $d_{i,u} \triangleright d_{i,v}$ 的文档对，以避免重复考虑同一个文档对。

8.4.2 IR-SVM 算法

在讨论逐点算法的缺点时，我们提到如果查询对应的文档集是不平衡的话，会导致那些文档数更多的查询更多地影响算法。在极端的情况下，很多对应少量文档的查询可能对所得的模型几乎没有影响，而模型的参数更多是由那些对应文档数多的查询决定。在逐对方法中，这个问题同样存在，而且还进一步恶化。在 Ranking SVM 中，对于查询 q_i 有 N_i 个不同的文档，那么满足条件 $d_{i,u} \triangleright d_{i,v}$ （即 $y_{i,(u,v)} = 1$ ）的文档对 (d_u, d_v) 的数目的数量级为 $O(N_i^2)$ 。在这种情况下，不平衡的情况会更加恶化，使得逐对方法中模型的学习会更加依赖那些对应文档数较多的查询。

为了解决查询间的不平衡问题，一种解决方案就是在求解优化问题时，将所得的经验损失函数对每个查询贡献的部分进行修正，使得经验损失函数不被那些对应文档数目较多的查询所支配。简而言之，就是对损失函数进行查询级别的标准化的（query level normalization）。具体而言，就是将损失函数中每个查询对应的项除以该查询所对应的文档对的数目。这样，我们就可以在 Ranking SVM 的基础上得到新的算法，称为 IR-SVM。将查询 q_i 所有满足条件 $d_{i,u} \triangleright d_{i,v}$ 的文档对 (d_u, d_v) 的数目记为 M_i ，在 IR-SVM 中，我们求解的优化问题如下所示：

$$\begin{aligned}
& \min \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^m \frac{\sum_{u,v: y_{i,(u,v)}=1} \varepsilon_{i,(u,v)}}{M_i} \\
& \text{s.t. } \mathbf{w}^T (\mathbf{x}_{i,u} - \mathbf{x}_{i,v}) \geq 1 - \varepsilon_{i,(u,v)}, \text{ 如果 } y_{i,(u,v)} = 1 \\
& \varepsilon_{i,(u,v)} \geq 0, \quad i = 1, 2, \dots, m, \quad d_{i,u} \succ d_{i,v} \in P_i
\end{aligned} \tag{8-15}$$

大量实验结果表明, IR-SVM 性能比 Ranking SVM 有了显著提高, 这就说明了在查询层次进行标准化处理的重要性。

8.4.3 RankNet 算法

RankNet 算法^[30]是较早工业界被实际部署的算法之一。在 RankNet 算法中, 我们考虑所有的文档对, 并使用交叉熵 (cross entropy) 损失函数来度量损失以最小化被错分的文档对数目。为了最小化交叉熵损失函数, RankNet 构建了一个人工神经网络, 并使用随机梯度下降算法来求解其中的参数。下面我们详细介绍 RankNet 算法, 并介绍其对应的损失函数。由于本书没有涉及人工神经网络相关的讨论, 因此省略该算法的具体实现, 主要讨论 RankNet 的原理及其中的关键步骤, 即求解导数。在 RankNet 算法的基础上, 我们进一步讨论 LambdaRank 和 LambdaMART 算法及其具体实现, 并介绍在 R 中如何使用 LambdaMART 算法。

对于查询 q_i 以及给定的文档对 $(d_{i,u}, d_{i,v})$, 我们用 $\bar{P}_{i,(u,v)}$ 表示在真实情况中, 查询 q_i 对应的文档 $d_{i,u}$ 排在 $d_{i,v}$ 之前的概率:

$$\bar{P}_{i,(u,v)} = \begin{cases} 1, & \text{如果 } d_{i,u} \succ d_{i,v} \\ 0, & \text{如果 } d_{i,v} \succ d_{i,u} \end{cases} \tag{8-16}$$

与前面讨论的算法类似, 我们使用类标 $y_{i,(u,v)}$ 来表示对于查询 q_i , 文档 $d_{i,u}$ 和 $d_{i,v}$ 之间的真实排序关系。

根据 $\bar{P}_{i,(u,v)}$ 和 $y_{i,(u,v)}$ 的定义, 有:

$$\bar{P}_{i,(u,v)} = \frac{1}{2} (1 + y_{i,(u,v)}) \tag{8-17}$$

这里我们将排序模型记为 f , 模型参数记为 \mathbf{w} 。在 RankNet 算法中, 采用的模型是人工神经网络, \mathbf{w} 就是人工神经网络的参数。这里为了书写方便, 我们将模型 f 对于 $\mathbf{x}_{i,u}$ 的输出 $f(\mathbf{x}_{i,u})$ 记为 $s_{i,u}$, 即 $s_{i,u} = f(\mathbf{x}_{i,u})$ 。利用 $f(\mathbf{x}_{i,u})$ 和 $f(\mathbf{x}_{i,v})$, 我们可以定义在给定模型 f 的情况下, 对于文档对 $(d_{i,u}, d_{i,v})$, 文档 $d_{i,u}$ 排在文档 $d_{i,v}$ 之前的概率 $P_{i,(u,v)}(f)$:

$$P_{i,(u,v)}(f) \equiv P(d_{i,u} \succ d_{i,v}) \equiv \text{sig}(\theta(s_{i,u} - s_{i,v})) = \frac{1}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} \tag{8-18}$$

这里我们使用了 sigmoid 函数 $\text{sig}(x) = \frac{1}{1 + \exp(-x)}$ 来将 $\theta(s_{i,u} - s_{i,v})$ 转化为概率, 其中 θ 是控制参数。这里概率的定义与逻辑回归类似。

根据前面的定义, 我们可以将交叉熵 (cross-entropy) 损失函数表示为:

$$L(f, \mathbf{x}_{i,u}, \mathbf{x}_{i,v}, y_{i,(u,v)}) = -\bar{P}_{i,(u,v)} \ln P_{i,(u,v)}(f) - (1 - \bar{P}_{i,(u,v)}) \ln (1 - P_{i,(u,v)}(f)) \quad (8-19)$$

注意, 这里我们只考虑了 q_i 、 $d_{i,u}$ 、 $d_{i,v}$ 。如果我们考虑了训练集中所有的 i 、 u 、 v 的组合, 则得到整个训练集对应的交叉熵损失函数。

接下来我们要将函数 f 代入损失函数中并简化。不失一般性, 这里我们将 $P_{i,(u,v)}(f)$ 写为 $P_{i,(u,v)}$ 。首先我们注意到

$$\ln P_{i,(u,v)} = \ln \frac{1}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} = -\ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) \quad (8-20)$$

$$\begin{aligned} \ln(1 - P_{i,(u,v)}) &= \ln \left(1 - \frac{1}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} \right) = \ln \frac{\exp(-\theta(s_{i,u} - s_{i,v}))}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} \\ &= -\theta(s_{i,u} - s_{i,v}) - \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) \end{aligned} \quad (8-21)$$

我们将 $L(f, \mathbf{x}_{i,u}, \mathbf{x}_{i,v}, y_{i,(u,v)})$ 简记为 $L_{i,(u,v)}$, 并简化如下:

$$\begin{aligned} L_{i,(u,v)} &= -\bar{P}_{i,(u,v)} \ln P_{i,(u,v)} - (1 - \bar{P}_{i,(u,v)}) \ln (1 - P_{i,(u,v)}) \\ &= \frac{1}{2}(1 + y_{i,(u,v)}) \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) \\ &\quad - \frac{1}{2}(1 - y_{i,(u,v)}) \left[-\theta(s_{i,u} - s_{i,v}) - \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) \right] \\ \Rightarrow 2L_{i,(u,v)} &= \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) + y_{i,(u,v)} \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) + \theta(s_{i,u} - s_{i,v}) \\ &\quad + \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) - y_{i,(u,v)} \theta(s_{i,u} - s_{i,v}) - y_{i,(u,v)} \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) \\ &= 2 \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) + (1 - y_{i,(u,v)}) \theta(s_{i,u} - s_{i,v}) \quad (8-22) \\ \Rightarrow L_{i,(u,v)} &= \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) + \frac{1}{2}(1 - y_{i,(u,v)}) \theta(s_{i,u} - s_{i,v}) \end{aligned}$$

注意, 当 $y_{i,(u,v)} = 1$ (即 $d_{i,u} > d_{i,v}$) 时, 有:

$$L_{i,(u,v)} = \ln(1 + \exp(-\theta(s_{i,u} - s_{i,v}))) \quad (8-23)$$

这样我们得到了 $L_{i,(u,v)}$ 关于 $s_{i,u}$ 、 $s_{i,v}$ 的表达式。根据该公式, 我们可以计算 $L_{i,(u,v)}$ 关于 $s_{i,u}$

的偏导数 $\frac{\partial L_{i,(u,v)}}{\partial s_{i,u}}$ 。事实上, 在 RankNet 算法中, 我们使用随机梯度下降算法来求解参数 \mathbf{w} 。

在推荐算法中, 我们已经介绍了随机梯度算法的基本原理, 这里我们直接予以应用。

在使用随机梯度下降算法求解模型参数 \mathbf{w} 时, 我们需要计算 $\frac{\partial L}{\partial \mathbf{w}}$, 即损失函数对于参数 \mathbf{w} 的偏导数。利用该偏导数 $\frac{\partial L}{\partial \mathbf{w}}$, 我们可以逐步更新模型的参数 \mathbf{w} 。在 RankNet 算法中, 我们使用神经网络来构建模型 f , 因此我们知道参数 \mathbf{w} 和模型输出 $s_{i,u} = f(\mathbf{x}_{i,u})$ 的关系, 且偏导数 $\frac{\partial s_{i,u}}{\partial \mathbf{w}}$ 也是比较容易计算的。这样我们可以利用链式规则得到 $\frac{\partial L}{\partial \mathbf{w}}$ 。在本书中, 我们省略了关于人工神经网络的讨论, 因此这里也不深入讨论 $\frac{\partial s_{i,u}}{\partial \mathbf{w}}$ 的计算了。我们主要讨论如何计算偏导数 $\frac{\partial L_{i,(u,v)}}{\partial s_{i,u}}$ 和 $\frac{\partial L_{i,(u,v)}}{\partial s_{i,v}}$, 其具体计算过程如下:

$$\begin{aligned}\frac{\partial L_{i,(u,v)}}{\partial s_{i,u}} &= \frac{1}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} \exp(-\theta(s_{i,u} - s_{i,v}))(-\theta) + \frac{1}{2}(1 - y_{i,(u,v)})\theta \\ &= \theta \left(\frac{-\exp(-\theta(s_{i,u} - s_{i,v}))}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} + \frac{1}{2}(1 - y_{i,(u,v)}) \right) \\ &= \theta \left(\frac{1}{2}(1 - y_{i,(u,v)}) - \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \right)\end{aligned}\quad (8-24)$$

类似地, 可以得到:

$$\begin{aligned}\frac{\partial L_{i,(u,v)}}{\partial s_{i,v}} &= \frac{1}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} \exp(-\theta(s_{i,u} - s_{i,v}))\theta - \frac{1}{2}(1 - y_{i,(u,v)})\theta \\ &= \theta \left(\frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} - \frac{1}{2}(1 - y_{i,(u,v)}) \right).\end{aligned}\quad (8-25)$$

因此,

$$\frac{\partial L_{i,(u,v)}}{\partial s_{i,u}} = -\frac{\partial L_{i,(u,v)}}{\partial s_{i,v}}\quad (8-26)$$

从式 (8-26) 可以看出, 当我们利用文档对 $d_{i,u} \triangleright d_{i,v}$ 来计算导数时, $\frac{\partial L_{i,(u,v)}}{\partial s_{i,u}}$ 与 $\frac{\partial L_{i,(u,v)}}{\partial s_{i,v}}$ 互

为相反数。

利用链式规则, 可以计算 $L_{i,(u,v)}$ 对模型参数 w 的偏导数:

$$\begin{aligned}
 \frac{\partial L_{i,(u,v)}}{\partial w} &= \frac{\partial L_{i,(u,v)}}{\partial s_{i,u}} \frac{\partial s_{i,u}}{\partial w} + \frac{\partial L_{i,(u,v)}}{\partial s_{i,v}} \frac{\partial s_{i,v}}{\partial w} = \frac{\partial L_{i,(u,v)}}{\partial s_{i,u}} \frac{\partial s_{i,u}}{\partial w} - \frac{\partial L_{i,(u,v)}}{\partial s_{i,v}} \frac{\partial s_{i,v}}{\partial w} \\
 &= \frac{\partial L_{i,(u,v)}}{\partial s_{i,u}} \left(\frac{\partial s_{i,u}}{\partial w} - \frac{\partial s_{i,v}}{\partial w} \right) \\
 &= \theta \left(\frac{1}{2} (1 - y_{i,(u,v)}) - \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \right) \left(\frac{\partial s_{i,u}}{\partial w} - \frac{\partial s_{i,v}}{\partial w} \right) \\
 &= \lambda_{i,(u,v)} \left(\frac{\partial s_{i,u}}{\partial w} - \frac{\partial s_{i,v}}{\partial w} \right)
 \end{aligned} \tag{8-27}$$

这里 $\lambda_{i,(u,v)}$ 定义为:

$$\lambda_{i,(u,v)} \equiv \theta \left(\frac{1}{2} (1 - y_{i,(u,v)}) - \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \right) \tag{8-28}$$

因此, 对于 $L_{i,(u,v)}$, 在随机梯度下降算法中, 我们可使用如下公式来更新模型的参数 w :

$$w \leftarrow w - \eta \frac{\partial L_{i,(u,v)}}{\partial w} = w - \eta \lambda_{i,(u,v)} \left(\frac{\partial s_{i,u}}{\partial w} - \frac{\partial s_{i,v}}{\partial w} \right) \tag{8-29}$$

这里 $\eta > 0$ 是随机梯度下降算法中的学习率 (learning rate)。

RankNet 算法的 Mini-batch 实现

上面的随机梯度下降算法对于训练集中每个查询对应的每个文档对都要更新一次模型参数 w , 在实际应用中效率很低。为了进一步提高算法的效率, RankNet 的作者提出了 Mini-batch 算法。其基本思想是将每个查询 q_i 对应的所有文档对放在一起考虑, 一并更新模型的参数 w 。

对于训练集中的所有文档对, 我们假设第一项应该排在第二项之前, 这样我们在讨论 $\lambda_{i,(u,v)}$ 时都有 $d_{i,u} \triangleright d_{i,v}$ 。对参数 w 的更新公式可以写为:

$$w \leftarrow w - \eta \sum_{(d_{i,u}, d_{i,v}) \in P_i} \frac{\partial L_{i,(u,v)}}{\partial w} = w - \eta \sum_{(d_{i,u}, d_{i,v}) \in P_i} \lambda_{i,(u,v)} \left(\frac{\partial s_{i,u}}{\partial w} - \frac{\partial s_{i,v}}{\partial w} \right) \tag{8-30}$$

接下来考虑简化累加项。考虑累加项中涉及 $\frac{\partial s_{i,u}}{\partial w}$ 的项, 分为两种情况: (1) 文档 $d_{i,u}$ 排在另一文档之前; (2) 文档 $d_{i,u}$ 排在另一文档之后。在第一种情况中, 如果文档 $d_{i,u}$ 排在另一文档 $d_{i,v}$ 之前, 其和为:

$$\sum_{(d_{i,u}, d_{i,v}) \in P_i} \lambda_{i,(u,v)} \frac{\partial s_{i,u}}{\partial w} \tag{8-31}$$

在第二种情况中, 如果文档 $d_{i,u}$ 排在另一文档 $d_{i,k}$ 之后, 其和为:

$$\sum_{(d_{i,k}, d_{i,u}) \in P_i} -\lambda_{i,(k,u)} \frac{\partial s_{i,u}}{\partial \mathbf{w}} = - \sum_{(d_{i,k}, d_{i,u}) \in P_i} \lambda_{i,(k,u)} \frac{\partial s_{i,u}}{\partial \mathbf{w}} \quad (8-32)$$

将最后的累加中所有涉及 $\lambda_{i,(u,v)}$ 的项记为 $\lambda_{i,u}$:

$$\lambda_{i,u} = \sum_{(d_{i,u}, d_{i,v}) \in P_i} \lambda_{i,(u,v)} - \sum_{(d_{i,k}, d_{i,u}) \in P_i} \lambda_{i,(k,u)} \quad (8-33)$$

对参数 \mathbf{w} 的更新公式可以简写为:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{d_{i,u} \in D_i} \lambda_{i,u} \frac{\partial s_{i,u}}{\partial \mathbf{w}} \quad (8-34)$$

利用式 (8-34), 我们对于查询 q_i 所对应的所有文档集合 D_i 中的每个文档 $d_{i,u}$ 计算相应的 $\lambda_{i,u}$, 并计算 $\sum_{d_{i,u} \in D_i} \lambda_{i,u} \frac{\partial s_{i,u}}{\partial \mathbf{w}}$, 之后只需要更新参数 \mathbf{w} 一次即可。注意, 这里的 $\frac{\partial s_{i,u}}{\partial \mathbf{w}}$ 与具体的模型有关系, 但是一般来讲较容易计算。这样的话, 对于每个查询, 我们在计算累加项后只需要更新一次参数即可。在 RankNet 中, 由于每次更新人工神经网络的参数比较耗费时间, 因此这种 Mini-batch 的随机梯度下降算法在实际计算中能够显著提高计算速度。在原始的随机梯度下降算法中, 如果对于每个文档对更新一下模型参数, 则训练时间是每个查询对应文档总数的二次函数; 而使用 Mini-batch 算法之后, 时间复杂度大致降为每个查询对应文档总数的线性函数。同时, Mini-batch 算法也是 LambdaRank 算法的基础, 下文会详细讨论。

8.4.4 LambdaRank 算法

在 RankNet 算法中, 我们直接优化了被错分的文档对的数目 (严格地讲, 只是被错分的文档对的数目的一个近似)。而在实际工作中, 往往需要直接优化我们前面讨论的那些适用于排序问题的评价指标, 如 NDCG。事实上, 直接最小化被错分的文档对的数目并不等于优化对应的指标。下面我们以 NDCG 为例, 给出一个具体的例子。

在这个例子中, 我们给出了一个查询的两个不同的排序结果, 如图 8-3 所示。在这个例子中, 返回的结果中一共有 16 个文档。其中, 深灰色代表对应的文档是真正和查询相关的, 而浅灰色的则代表事实上和查询不相关的文档。在图 8-3 (a) 所示的排序结果中, 第 1 个和第 15 个文档是和查询真正相关的, 因此被错分的文档对的数目是 13; 在图 8-3 (b) 所示的排序结果中, 第 4 个和第 10 个文档是和查询真正相关的, 因此被错分的文档对的数目是 11。但通过简单计算 NDCG 可以知道, 这两个排序结果对应的 NDCG 是不同的。在 NDCG 中, 我们更加重视返回的文档序列中最前面的结果, 因此图 8-3 (a) 的例子对应的 NDCG 更高。从直观上讲, 在实际中我们更希望排序算法返回的最前面的结果是与用户查询相关的, 因此这个例子也显示了直接优化被错分的文档对的数目在实际中并不一定是最优的选择。

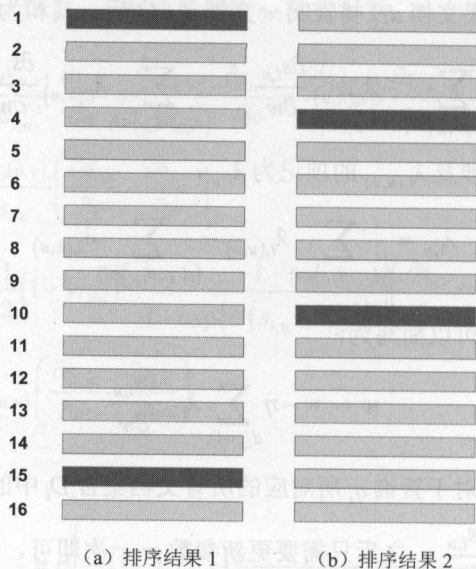


图 8-3 不同排序算法评价标准的比较

为了直接、有效地优化 NDCG, 在 RankNet 算法的基础上, 原作者又提出了 LambdaRank 算法^[31]。注意, 在 RankNet 中采用随机梯度下降算法求解问题时, 每一步只需要根据训练数计算梯度值并更新模型参数。在 LambdaRank 算法中, 核心还是利用梯度信息来更新模型的参数, 但是最重要的一点是直接在梯度中引入了 NDCG 相关的信息。

在前面的 RankNet 算法中, 我们在计算梯度的过程中引入了 $\lambda_{i,(u,v)}$ 。回忆一下, 在 RankNet 算法中 $\lambda_{i,(u,v)}$ 可以使用如下公式计算:

$$\lambda_{i,(u,v)} = \frac{\partial L_{i,(u,v)}}{\partial s_{i,u}} = \theta \left(\frac{1}{2} (1 - y_{i,(u,v)}) - \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \right) \quad (8-35)$$

对于训练集中的所有文档对, 我们假设第一项应该排在第二项之前, 即 $d_{i,u} \triangleright d_{i,v}$ 。这样的话, 我们直接可以得到 $y_{i,(u,v)} = 1$ 。这样可以将 $\lambda_{i,(u,v)}$ 的计算公式简写为:

$$\lambda_{i,(u,v)} = \frac{-\theta}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \quad (8-36)$$

在使用随机梯度算法求解 RankNet 算法时, 我们可以将 $\lambda_{i,(u,v)}$ 视为从文档对 $d_{i,u} \triangleright d_{i,v}$ 提取的有用信息。在实际中, 人们发现, 如果在上面的 $\lambda_{i,(u,v)}$ 中简单乘以 NDCG 的变化量 (记为 $|\Delta_{i,(u,v)}^{NDCG}|$), 可以得到更好的排序结果。这里 $|\Delta_{i,(u,v)}^{NDCG}|$ 是我们将当前模型 f 所得排序结果中文档 $d_{i,u}$ 和 $d_{i,v}$ 交换位置 (保持其他文档的排序位置不变) 所引起的 NDCG 的变化的绝对值。这样, 在 LambdaRank 算法中, 我们使用如下公式来计算新的 $\lambda_{i,(u,v)}$, 记为 $\lambda_{i,(u,v)}^{LR}$:

$$\lambda_{i,(u,v)}^{LR} = \frac{-\theta}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \left| \Delta_{i,(u,v)}^{NDCG} \right| \quad (8-37)$$

在 $\lambda_{i,(u,v)}^{LR}$ 的基础上, 可以认为存在一个新的损失函数 L^{LR} , 使得

$$\lambda_{i,(u,v)}^{LR} \equiv \frac{\partial L_{i,(u,v)}^{LR}}{\partial s_{i,u}} \quad (8-38)$$

这里 $L_{i,(u,v)}^{LR}$ 是 L^{LR} 对应于文档对 $d_{i,u} \triangleright d_{i,v}$ 的部分。由于在 LambdaRank 算法中我们仅仅改变了梯度的计算, 因此仍然可以采用随机梯度下降算法来优化模型的参数。具体来说, 就是使用如下公式来更新模型的参数 \mathbf{w} :

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial L_{i,(u,v)}^{LR}}{\partial \mathbf{w}} = \mathbf{w} - \eta \lambda_{i,(u,v)}^{LR} \left(\frac{\partial s_{i,u}}{\partial \mathbf{w}} - \frac{\partial s_{i,v}}{\partial \mathbf{w}} \right) \quad (8-39)$$

类似地, 我们也可以推导出 LambdaRank 中的对应的 Mini-batch 算法。将文档 $d_{i,u}$ 所有对应的 $\lambda_{i,(u,v)}^{LR}$ 累加在一起, 可以得到 $\lambda_{i,u}^{LR}$:

$$\begin{aligned} \lambda_{i,u}^{LR} &= \sum_{(d_{i,u}, d_{i,v}) \in P_i} \lambda_{i,(u,v)}^{LR} - \sum_{(d_{i,k}, d_{i,u}) \in P_i} \lambda_{i,(k,u)}^{LR} \\ &= \sum_{(d_{i,u}, d_{i,v}) \in P_i} \frac{-\theta}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \left| \Delta_{i,(u,v)}^{NDCG} \right| - \sum_{(d_{i,k}, d_{i,u}) \in P_i} \frac{\theta}{1 + \exp(\theta(s_{i,k} - s_{i,u}))} \left| \Delta_{i,(k,u)}^{NDCG} \right| \\ &= -\theta \left(\sum_{(d_{i,u}, d_{i,v}) \in P_i} \left| \Delta_{i,(u,v)}^{NDCG} \right| \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} - \sum_{(d_{i,k}, d_{i,u}) \in P_i} \left| \Delta_{i,(k,u)}^{NDCG} \right| \frac{1}{1 + \exp(\theta(s_{i,k} - s_{i,u}))} \right) \end{aligned} \quad (8-40)$$

这样, 使用 $\lambda_{i,u}^{LR}$ 就综合考虑了当查询为 q_i 时所有涉及文档 $d_{i,u}$ 的文档对相关的信息。使用 $\lambda_{i,u}^{LR}$ 我们就得到了 LambdaRank 算法对应的 Mini-batch 算法。其中, 参数 \mathbf{w} 的更新公式可以简写为:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{d_{i,u} \in D_i} \lambda_{i,u}^{LR} \frac{\partial s_{i,u}}{\partial \mathbf{w}} \quad (8-41)$$

8.4.5 LambdaMART 算法

我们知道, LambdaRank 算法基于 RankNet, 而 LambdaMART 算法^[32]可以认为是将梯度提升算法和 LambdaRank 算法结合的产物。LambdaMART 算法多次在机器学习竞赛中取得优异的成绩, 如 2010 年的 Yahoo! Learning to Rank Challenge, 以及 2013 年由 ICDM 会议和 Expedia 公司共同组织的竞赛^①, 获胜方案都使用了 LambdaMART。

① 在 1.3.4 节中我们对相关问题进行了讨论。

在 LambdaMART 算法中, 我们利用 LambdaRank 算法中的梯度来构造对应的损失函数。在给定损失函数之后, 我们使用提升决策树 (boosted tree), 即一系列决策树的加权和作为最终模型。在提升决策树中, 我们顺次构建多个决策树, 希望新的决策树能够弥补已有的决策树的“不足”。在实现上, 我们使用梯度提升算法来学习这些决策树。梯度提升算法的基本思想与数值优化中的梯度下降算法类似。在梯度提升算法中, 我们每次都根据已有的模型的预测结果, 构建新的目标值以更好地分类那些“难分类”的样本。具体来说, 对于样本 \mathbf{x}_i , 在构建第 j 个学习器时, 我们用 t_{ij} 来表示对应的目标值。通过不停地改变 t_{ij} , 使得综合后的学习器能够更好地处理样本 \mathbf{x}_i 。在梯度提升算法中, t_{ij} 是损失函数关于当前模型预测值的负梯度。关于提升树和梯度提升算法的具体介绍可参见第 9 章相关内容。在第 9 章中, 我们将详细介绍梯度提升算法。在本节, 我们着重讨论如何计算梯度和步长。读者可以先阅读第 9 章的相关章节再阅读本节。

可以简单地理解, 在 LambdaMART 算法中, 对于查询 q_i , 我们考虑的损失函数为:

$$L_i^{LM} = \sum_{(d_{i,u}, d_{i,v}) \in P_i} \left| \Delta Z_{i,(u,v)} \right| \ln \left(1 + \exp \left(-\theta (s_{i,u} - s_{i,v}) \right) \right) = \sum_{(d_{i,u}, d_{i,v}) \in P_i} \left| \Delta Z_{i,(u,v)} \right| \Phi_{i,(u,v)} \quad (8-42)$$

式中 $\Phi_{i,(u,v)}$ 定义为:

$$\Phi_{i,(u,v)} = \ln \left(1 + \exp \left(-\theta (s_{i,u} - s_{i,v}) \right) \right) \quad (8-43)$$

这里 $\Delta Z_{i,(u,v)}$ 是我们将文档 $d_{i,u}$ 和 $d_{i,v}$ 交换位置 (而其他文档位置不变) 后所引起的评价指标的变化。例如, 采用 NDCG 时, $\Delta Z_{i,(u,v)} = \Delta_{i,(u,v)}^{NDCG}$ 。此外, 与前面的假设相同, 对于训练集中的所有文档对, 我们假设第一项应该排在第二项之前, 即 $d_{i,u} \triangleright d_{i,v}$ 。

将所有的查询都考虑, 则得到最终的损失函数:

$$L^{LM} = \sum_{i=1}^m L_i^{LM} = \sum_{i=1}^m \sum_{d_{i,u} \triangleright d_{i,v}} \left| \Delta Z_{i,(u,v)} \right| \Phi_{i,(u,v)} \quad (8-44)$$

可以看出, L_i^{LM} 与前面讨论的 RankNet 算法和 LambdaRank 算法都有一定的相似之处。事实上, 如果我们进一步计算 L_i^{LM} 对于当前评分 $s_{i,u}$ 的一阶导数和二阶导数的话, 它们之间的联系会更加清楚。

首先我们计算 $\Phi_{i,(u,v)}$ 对 $s_{i,u}$ 、 $s_{i,v}$ 的偏导数:

$$\begin{aligned} \frac{\partial \Phi_{i,(u,v)}}{\partial s_{i,u}} &= \frac{1}{1 + \exp \left(-\theta (s_{i,u} - s_{i,v}) \right)} \exp \left(-\theta (s_{i,u} - s_{i,v}) \right) (-\theta) \\ &= -\theta \frac{\exp \left(-\theta (s_{i,u} - s_{i,v}) \right)}{1 + \exp \left(-\theta (s_{i,u} - s_{i,v}) \right)} \\ &= -\theta \frac{1}{1 + \exp \left(\theta (s_{i,u} - s_{i,v}) \right)} = -\theta \rho_{i,uv} \end{aligned} \quad (8-45)$$

$$\begin{aligned}
\frac{\partial \Phi_{i,(u,v)}}{\partial s_{i,v}} &= \frac{1}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} \exp(-\theta(s_{i,u} - s_{i,v})) \theta \\
&= \theta \frac{\exp(-\theta(s_{i,u} - s_{i,v}))}{1 + \exp(-\theta(s_{i,u} - s_{i,v}))} \\
&= \theta \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} = \theta \rho_{i,uv}
\end{aligned} \tag{8-46}$$

这里记 $\rho_{i,uv}$ 为:

$$\rho_{i,uv} = \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \tag{8-47}$$

接下来我们求解 L_i^{LM} 对于当前评分 $s_{i,u}$ 的一阶导数。注意, 对于文档 $d_{i,u}$, 我们要考虑在真实数据的文档对中, 文档 $d_{i,u}$ 排在另一文档之前还是之后。这样, 我们有:

$$\begin{aligned}
\frac{\partial L_i^{LM}}{\partial s_{i,u}} &= \sum_{(d_{i,u}, d_{i,v}) \in P_i} |\Delta Z_{i,(u,v)}| (-\theta \rho_{i,uv}) + \sum_{(d_{i,k}, d_{i,u}) \in P_i} |\Delta Z_{i,(k,u)}| (\theta \rho_{i,ku}) \\
&= -\theta \left(\sum_{(d_{i,u}, d_{i,v}) \in P_i} |\Delta Z_{i,(u,v)}| \rho_{i,uv} - \sum_{(d_{i,k}, d_{i,u}) \in P_i} |\Delta Z_{i,(k,u)}| \rho_{i,ku} \right)
\end{aligned} \tag{8-48}$$

比较 $\frac{\partial L_i^{LM}}{\partial s_{i,u}}$ 和前面 LambdaRank 算法中的 $\lambda_{i,u}^{LR}$, 将上式中的 $|\Delta Z_{i,(u,v)}|$ 替换为 $|\Delta_{i,(u,v)}^{NDCG}|$, 我们可以发现 $\frac{\partial L_i^{LM}}{\partial s_{i,u}} = \lambda_{i,u}^{LR}$ 。换言之, 在 LambdaMART 算法中, 我们使用了与 LambdaRank 算法中

类似的导数来处理不同的评价标准。这里我们也把 $\frac{\partial L_i^{LM}}{\partial s_{i,u}}$ 记为 $\lambda_{i,u}^{LM} = \frac{\partial L_i^{LM}}{\partial s_{i,u}}$ 。

在这里讨论的梯度提升算法中, 我们使用牛顿近似 (即二阶泰勒展式) 来拟合函数 L_i^{LM} 以求步长从而加速收敛 (具体原理可参见 9.5.1 节)。在二阶泰勒展式中, 我们需要计算一阶导数 $\frac{\partial L_i^{LM}}{\partial s_{i,u}}$ 和二阶导数 $\frac{\partial^2 L_i^{LM}}{\partial s_{i,u}^2}$ 。前面我们已经给出了计算 $\frac{\partial L_i^{LM}}{\partial s_{i,u}}$ 的公式, 下面给出计算 $\frac{\partial^2 L_i^{LM}}{\partial s_{i,u}^2}$ 的具体公式。我们首先计算 $\rho_{i,uv}$ 对 $s_{i,u}$ 和 $s_{i,v}$ 的导数:

$$\begin{aligned}
\frac{\partial \rho_{i,uv}}{\partial s_{i,u}} &= \frac{-\exp(\theta(s_{i,u} - s_{i,v})) \theta}{(1 + \exp(\theta(s_{i,u} - s_{i,v})))^2} = -\theta \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \frac{\exp(\theta(s_{i,u} - s_{i,v}))}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \\
&= -\theta \rho_{i,uv} (1 - \rho_{i,uv})
\end{aligned} \tag{8-49}$$

$$\frac{\partial \rho_{i,uv}}{\partial s_{i,v}} = \frac{\exp(\theta(s_{i,u} - s_{i,v}))\theta}{\left(1 + \exp(\theta(s_{i,u} - s_{i,v}))\right)^2} = \theta \frac{1}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \frac{\exp(\theta(s_{i,u} - s_{i,v}))}{1 + \exp(\theta(s_{i,u} - s_{i,v}))} \quad (8-50)$$

$$= \theta \rho_{i,uv} (1 - \rho_{i,uv})$$

因此, 可以计算二阶导数 $\frac{\partial^2 L_i^{LM}}{\partial s_{i,u}^2}$ 如下:

$$\begin{aligned} \frac{\partial^2 L_i^{LM}}{\partial s_{i,u}^2} &= -\theta \left(\sum_{(d_{i,u}, d_{i,v}) \in P_i} |\Delta Z_{i,(u,v)}| \frac{\partial \rho_{i,uv}}{\partial s_{i,u}} - \sum_{(d_{i,k}, d_{i,u}) \in P_i} |\Delta Z_{i,(k,u)}| \frac{\partial \rho_{i,ku}}{\partial s_{i,u}} \right) \\ &= -\theta \left(\sum_{(d_{i,u}, d_{i,v}) \in P_i} |\Delta Z_{i,(u,v)}| (-\theta \rho_{i,uv} (1 - \rho_{i,uv})) - \sum_{(d_{i,k}, d_{i,u}) \in P_i} |\Delta Z_{i,(k,u)}| \theta \rho_{i,ku} (1 - \rho_{i,ku}) \right) \quad (8-51) \\ &= \theta^2 \left(\sum_{(d_{i,u}, d_{i,v}) \in P_i} |\Delta Z_{i,(u,v)}| \rho_{i,uv} (1 - \rho_{i,uv}) + \sum_{(d_{i,k}, d_{i,u}) \in P_i} |\Delta Z_{i,(k,u)}| \rho_{i,ku} (1 - \rho_{i,ku}) \right) \end{aligned}$$

在得到一阶导数和二阶导数之后, 在梯度提升算法中我们可以计算相应的步长。在构建的第 k 棵决策树对应的第 l 个叶结点时, 考虑该结点的数据 $\mathbf{x}_{i,u}$, 我们可以使用如下公式来计算采用牛顿近似时的步长 γ_{kl} :

$$\gamma_{kl} = - \frac{\sum_{\mathbf{x}_{i,u} \in R_{kl}} \frac{\partial L_i^{LM}}{\partial s_{i,u}}}{\sum_{\mathbf{x}_{i,u} \in R_{kl}} \frac{\partial^2 L_i^{LM}}{\partial s_{i,u}^2}} \quad (8-52)$$

这里我们使用 R_{kl} 表示第 k 棵决策树对应的第 l 个叶结点所对应的数据的集合。

在 LambdaMART 算法中, 我们使用梯度提升算法来构建模型, 具体步骤见算法 8-1。

算法 8-1 LambdaMART 算法

输入: 决策树的总数 N_t , 每棵树可允许的最大叶结点数 L_{\max} , 学习率 η , 总查询数 m

具体步骤:

for all $\mathbf{x}_{i,u}$ do

$f_0(\mathbf{x}_{i,u}) = \text{BaseModel}(\mathbf{x}_{i,u})$

end for

for $k = 1:N_t$ do

for all $\mathbf{x}_{i,u}$ do

计算梯度 $\lambda_{i,u}^{LM}$: $\lambda_{i,u}^{LM} = -\theta(\sum_{(d_{i,u}, d_{i,v}) \in P_i} |\Delta Z_{i,(u,v)}| \rho_{i,uv} - \sum_{(d_{i,k}, d_{i,u}) \in P_i} |\Delta Z_{i,(k,u)}| \rho_{i,ku})$

将 $\lambda_{i,u}^{LM}$ 作为第 i 个样本的新的目标值 $y_{i,u} : y_{i,u} = \lambda_{i,u}^{LM}$

计算二阶导数 $\frac{\partial^2 L_i^{LM}}{\partial s_{i,u}^2}$

end for

根据训练数据 $\{x_{i,u}, y_{i,u}\}$ 构建一个有 L_{\max} 个叶结点的决策树

for $l = 1:L_{\max}$ do

为第 l 个叶结点计算牛顿步长 γ_{kl} : $\gamma_{kl} = -\frac{\sum_{x_{i,u} \in R_{kl}} y_{i,u}}{\sum_{x_{i,u} \in R_{kl}} \frac{\partial^2 L_i^{LM}}{\partial s_{i,u}^2}}$

end for

for all $x_{i,u}$ do

更新函数 f : $f_k(x_{i,u}) = f_{k-1}(x_{i,u}) + \eta \sum_l \gamma_{kl} I(x_{i,u} \in R_{kl})$

end for

end for

在该算法中,我们首先建立一个基本模型 $BaseModel(x_{i,u})$ 。如果在实际中没有基本模型,可以直接设置 $BaseModel(x_{i,u}) = 0$ 。注意,算法中的 $I()$ 函数可参见式 (9-51)。

这里我们再简单讨论一下 LambdaRank 算法和 LambdaMART 算法的区别。首先它们使用的模型不同。在 LambdaRank 算法中我们使用人工神经网络,而在 LambdaMART 算法中我们使用一系列决策树。其次,在具体的算法中,模型中参数更新的方式也不一样。在 LambdaRank 算法中,我们每次处理查询 q_i 对应的一个文档对或者所有文档对。根据这部分数据,我们使用随机梯度下降算法来更新模型中所有的参数。而在 LambdaMART 算法中,我们使用梯度提升算法通过生成一系列决策树来构建模型。在每棵新决策树的生成过程中,我们使用对应于每个结点的所有数据来决定下面的结点如何生成。因此,一般而言,在决策树的一个结点的进一步划分过程中,我们可以使用该结点所对应的文档对。并且在这里我们只是考虑当前决策树的当前结点如何进一步划分,而不用考虑其他决策树。

gbm 包和 LambdaMART 算法

在 R 中,gbm 包^①提供了 LambdaMART 算法的实现。我们会在第 9 章中详细介绍 gbm 包的使用方法。这里我们简单讨论一下如何使用 gbm 包中的 LambdaMART 算法。

使用 gbm 包时,我们需要指定如下参数:

- 损失函数 (参数 distribution);
- 决策树的数目 (参数 n.trees);
- 决策树的内结点的数目 (参数 interaction.depth)。注意,指定每棵决策树中内结点的数目时,也决定了叶结点的最大数目;
- 学习率 (参数 shrinkage)。

① <https://cran.r-project.org/web/packages/gbm/index.html>

上面的参数是最重要的几个参数。除此之外，还有如下参数。

- 子取样的比例（参数 bag.fraction）。
- 划分结点时要求新的叶结点所拥有的最少样本数（参数 n.minobsinnode）。
- 训练时使用的数据比例（参数 train.fraction）。

在 gbm 包中，我们需要通过设置参数 'distribution' 来指定相应的损失函数。如果要在 gbm 中使用 LambdaMART 算法，则需要将 'distribution' 设为列表形式：

```
list(name="pairwise",group=...,metric=...,max.rank=...)
```

这里的 metric 是我们选择的排序算法性能的评价指标。在 gbm 包中，我们可以选择：

- NDCG;
- MAP;
- MRR (Mean Reciprocal Rank)，注意这里的 MRR 是排序最高的正类对象所对应的 MRR。
- Concordance：考察所有的文档对，Concordance 返回模型能够正确处理的文档对的比例。实际上就是在文档对分类问题上的 AUC。

在使用 LambdaMART 算法时，gbm 要求输入数据是数据框。在输入的数据框中，有一列表示该行数据关于其所对应的查询的信息，如查询的 id。这里我们使用 group 参数指定数据框中的哪一列或者哪些列对应查询。

由于 NDCG 在排序问题中的广泛使用，因此，在 gbm 实现的 LambdaMART 算法中，NDCG 是默认的评价指标。注意，在 gbm 包的 LambdaMART 算法的实现中，在使用 NDCG 和 Concordance 标准时，输入数据对应的目标值可以是任何值；但是当使用 MAP 和 MRR 时，目标值只能是 0 或者 1。

在计算 NDCG 和 MRR 时，我们可以设置参数 max.rank，这样就只考虑排序结果最前面的 max.rank 个返回结果。如果 max.rank 不设置，则整个返回结果都考虑。

下面这段程序显示了我们如何使用 gbm 来优化 NDCG。

```
gbm.ndcg <- gbm(Y~X1+X2+X3,           # formula
               data=data.train,        # dataset
               distribution=list(      # loss function:
                 name='pairwise',      # pairwise
                 metric="ndcg",       # ranking metric: NDCG
                 group='query'),      # column indicating query groups
               n.trees=2000,           # number of trees
               interaction.depth=3     # number of internal nodes
               shrinkage=0.005,        # learning rate
               bag.fraction = 0.5,     # subsampling fraction
               train.fraction = 1,     # fraction of data for training
               n.minobsinnode = 10,    # minimum number of obs for split
               keep.data=TRUE,         # store copy of input data in model
               cv.folds=5)             # number of cross validation folds
```

在这个例子中，我们假设输入数据保存在数据框 `data.train` 中，并且我们希望使用列 `X1`、`X2`、`X3` 来预测 `Y`，同时我们指定列 `query` 来表示查询。具体来说，就是将 `distribution` 参数设置为如下形式来使用 `gbm` 实现的 `LambdaMART` 算法：

```
distribution=list(name='pairwise', metric="ndcg", group='query')
```

在本书附带的 R 程序 `gbm_ranking_example.R` 中，我们详细介绍了如何使用 `gbm` 来优化 `NDCG` 和其他指标，感兴趣的读者可以直接运行该程序。

8.5 逐列方法

在逐列方法中，我们考察排序模型返回的整个序列。根据逐列方法的具体实现，基本上可分为两类。在第一类中，我们要直接输出每个查询和对应的文档之间的相关度。在输出的相关度的基础上，我们得到模型对于文档的排序结果，并比较真实的排序结果。通常我们定义损失函数并优化模型。一般来讲，这里的损失函数都是前面所讨论的关于排序算法的指标的近似函数或者上界。在第二类逐列方法中，对于每个查询，我们直接输出文档的排序结果序列，然后将该排序结果与真实排序进行比对，计算相应的损失函数并优化模型。注意，在第二类方法中，我们不需要知道查询和对应文档具体的相似度，只需要知道每个查询模型输出的排序结果即可。

为什么这里的损失函数都是评价标准的近似函数或者上界呢？原因在于排序算法中的评价标准（如 `NDCG` 和 `MAP`）都涉及排序结果中每个文档的具体位置，这些都是不连续而且不可导的，直接优化的求解难度很大。一般来说，在机器学习中，我们使用的损失函数都是连续且可导的，这样就可以使用数值优化中的很多优化算法来最小化损失函数。

与逐点和逐对方法不同，逐列方法直接从所得模型的排序结果入手。换言之，逐列方法考虑了排序模型对于每个查询 q_i 返回的整个排序序列并优化。与逐点方法相比，逐列方法和逐对方法类似，也是对每个查询单独考虑。比较常见的逐列方法包括 `ListNet`、`ListMLE`、`AdaRank`、`SVMmap`、`Soft Rank` 等。在本节中，我们着重介绍逐列方法的基本思想。在前面章节中，我们讨论了 `SVM` 在逐点和逐对方法中的扩展，在本节我们着重介绍 `SVM` 在逐列方法中的扩展 `SVMmap` 算法，这样有利于读者比较 `SVM` 算法在不同方法中的不同扩展形式。

8.5.1 SVM^{map} 算法

在 `LambdaMART` 算法中，我们以 `NDCG` 为例来说明如何优化排序算法中的评价指标。在 `SVMmap` 算法中，我们直接优化 `MAP`^①。具体来说，我们考虑排序模型输出的排序文档的整个序列。将输出的文档序列与真实的排序序列相比较并优化其差值，从而优化 `MAP`。

在具体介绍算法的细节之前，我们首先介绍一下在 `SVMmap` 算法中要用到的符号和基本

① `SVMmap` 算法可以修改，使之能够优化 `NDCG` 等其他标准，但是在本节中我们主要讨论如何优化 `MAP`。

假设。对于查询 q_i 和对应的文档 $d_{i,j}$ ，我们首先构建相应的特征向量 $\mathbf{x}_{i,j}$ 。为了简化讨论，我们假设相关度为 0 或者 1。对于模型的输出，我们通过计算 MAP 来衡量模型的好坏。与前面讨论的 SVM 模型的变体类似，我们假设要求一个线性模型 $f(\mathbf{x}_{i,j})$

$$f(\mathbf{x}_{i,j}) = \mathbf{w}^T \mathbf{x}_{i,j} \quad (8-53)$$

使得通过线性变换之后 MAP 最大。换言之，我们对特征 $\mathbf{x}_{i,j}$ 求解一个投影 \mathbf{w} ，使得投影后所得的 MAP 最大。

首先我们讨论查询 q_i 。将与 q_i 对应的文档 $d_{i,1}, d_{i,2}, \dots, d_{i,N_i}$ 中与 q_i 相关的文档的集合记为 D_i^r ，将其中与 q_i 不相关的文档集合记为 D_i^c 。从直观上讲，在投影后，如果 D_i^r 中的文档 $d_{i,u}$ 对应的 $\mathbf{w}^T \mathbf{x}_{i,u}$ 比 D_i^c 中的文档 $d_{i,v}$ 对应的 $\mathbf{w}^T \mathbf{x}_{i,v}$ 大，我们就能将两类文档分开。进一步分析，我们希望投影后我们选取的某项标准能够最大。在 SVM^{map} 算法中，我们希望 MAP 最大。

对于查询 q_i 的一个给定的排序结果 π ，我们定义一个函数 $F(\mathbf{x}_i, \pi; \mathbf{w})$ 来描述投影 \mathbf{w} 的好坏：

$$F(\mathbf{x}_i, \pi; \mathbf{w}) = \mathbf{w}^T \Psi(\mathbf{x}_i, \pi) \quad (8-54)$$

这里我们用 \mathbf{x}_i 表示查询 q_i 代表的所有输入数据，函数 $\Psi(\mathbf{x}_i, \pi)$ 的定义如下：

$$\Psi(\mathbf{x}_i, \pi) = \frac{1}{|D_i^r| |D_i^c|} \sum_{u: d_{i,u} \in D_i^r} \sum_{v: d_{i,v} \in D_i^c} s_{i,uv}(\pi) (\mathbf{x}_{i,u} - \mathbf{x}_{i,v}) \quad (8-55)$$

在排序 π 中，如果文档 $d_{i,u}$ 排在文档 $d_{i,v}$ 之前，则 $s_{i,uv}(\pi) = 1$ ；如果文档 $d_{i,u}$ 排在文档 $d_{i,v}$ 之后，则 $s_{i,uv}(\pi) = -1$ 。注意，文档 $d_{i,u}$ 是事实上与查询 q_i 相关的文档，而文档 $d_{i,v}$ 是与查询 q_i 不相关的文档。我们可以将函数 $F(\mathbf{x}_i, \pi; \mathbf{w})$ 进一步表示为

$$F(\mathbf{x}_i, \pi; \mathbf{w}) = \frac{1}{|D_i^r| |D_i^c|} \sum_{u: d_{i,u} \in D_i^r} \sum_{v: d_{i,v} \in D_i^c} s_{i,uv}(\pi) \mathbf{w}^T (\mathbf{x}_{i,u} - \mathbf{x}_{i,v}) \quad (8-56)$$

根据函数 $F(\mathbf{x}_i, \pi; \mathbf{w})$ 的定义，我们知道 $F(\mathbf{x}_i, \pi; \mathbf{w})$ 实际计算的是所有(相关，不相关)文档对根据在 π 中的排序位置在投影之后差的平均值。直观地讲，对于一个好的投影 \mathbf{w} ， $F(\mathbf{x}_i, \pi; \mathbf{w})$ 的值越大越好。与前面讨论的 LambdaRank 算法和 LambdaMART 算法类似，问题的难点是如何将 MAP 引入到优化问题中。

在 SVM^{map} 算法中，我们将错误排序 π 对应的 AP 值 $AP(\pi)$ 引入到优化问题的约束条件中，从而达到最优化 MAP 的目的。具体而言，我们求解如下的优化问题：

$$\min_{\mathbf{w}, \varepsilon_1, \varepsilon_2, \dots, \varepsilon_m} \frac{1}{2} \|\mathbf{w}\|_2^2 + \frac{C}{m} \sum_{i=1}^m \varepsilon_i \quad (8-57)$$

$$\text{s.t. } \forall i, \forall \pi \in \Pi_i \setminus \pi_i : F(\mathbf{x}_i, \pi_i; \mathbf{w}) \geq F(\mathbf{x}_i, \pi; \mathbf{w}) + (1 - AP(\pi)) - \varepsilon_i, \quad \varepsilon_i \geq 0$$

在这个优化问题中, 我们考虑了所有的 m 个查询。这里 Π_i 是 q_i 对应的文档集 $d_{i,1}, d_{i,2}, \dots, d_{i,N_i}$ 所有排序结果的集合, π_i 是 q_i 对应的文档集的真实排序, $C > 0$ 是控制参数, ε_i 是松弛变量。在这个优化问题的目标函数中, $\|\mathbf{w}\|_2^2$ 对应模型的复杂度, 第二项中 $\sum_{i=1}^m \varepsilon_i$ 则对应于由 MAP 引起的 Hinge 损失函数, $C > 0$ 则是控制第一项和第二项权重的参数。 $\Pi_i \setminus \pi_i$ 是对于 q_i 所有错误排序结果的集合, $AP(\pi)$ 表示对于查询 q_i 在真实排序是 π_i 的情况下, 当前排序为 π 所对应的 AP。因此, $1 - AP(\pi)$ 就是在真实排序是 π_i 的情况下当前排序为 π 时所带来的关于 MAP 的损失。

在上面的优化问题中, 可以将约束条件写为:

$$\forall i, \forall \pi \in \Pi_i \setminus \pi_i : \varepsilon_i \geq (F(x_i, \pi; \mathbf{w}) - F(x_i, \pi_i; \mathbf{w})) + (1 - AP(\pi)), \quad \varepsilon_i \geq 0 \quad (8-58)$$

换言之

$$\varepsilon_i = \max(0, (F(x_i, \pi; \mathbf{w}) - F(x_i, \pi_i; \mathbf{w})) + (1 - AP(\pi))) \quad (8-59)$$

在理想情况下, 我们希望这里错误的排序 π 与真实排序 π_i 越接近越好。这就意味着: 第一, 我们希望 $AP(\pi)$ 尽量大, 即 $1 - AP(\pi)$ 尽量小; 第二, 我们希望 $F(x_i, \pi; \mathbf{w}) - F(x_i, \pi_i; \mathbf{w})$ 越小越好。根据前面的讨论, 好的排序对应更大的 $F(x_i, \pi; \mathbf{w})$ 。在理想情况下, 真实的排序 π_i 对应的 $F(x_i, \pi_i; \mathbf{w})$ 值比所有错误的排序 π 对应的 $F(x_i, \pi; \mathbf{w})$ 值都高。如果一个错误的排序 π 对应的 $F(x_i, \pi; \mathbf{w})$ 值比真实的排序 π_i 对应的 $F(x_i, \pi_i; \mathbf{w})$ 值更高的话, 说明当前的 \mathbf{w} 值不理想。在这种情况下, 我们引入对应的惩罚项 $\varepsilon_i \geq 0$ 。

在约束条件中, 如果 $\pi = \pi_i$, 则 $\varepsilon_i = 0$; 如果错误排序 π 很糟糕, 则 $1 - AP(\pi)$ 很大, 或者, 如果 $F(x_i, \pi_i; \mathbf{w}) > F(x_i, \pi; \mathbf{w})$, 则会导致 ε_i 很大。因此, 在这个优化问题中, 我们最小化惩罚项 ε_i 之和。事实上, 我们可以严格地证明 $\frac{1}{m} \sum_{i=1}^m \varepsilon_i$ 是由 MAP 导致的损失函数的上界, 有兴趣

的读者可以参考相关文献[33]。

注意, 对于训练集中的每个查询对应的数据, 对于每种可能的错误排序结果, 我们都有一系列的约束条件。我们知道对于查询 q_i 相应的文档 $d_{i,1}, d_{i,2}, \dots, d_{i,N_i}$, 错误的排序结果的数目是 N_i 的指数级, 因此在前面的优化问题中, 我们也有 N_i 指数级的约束条件, 而且这还只是涉及查询 q_i 相关的部分。

为了有效地求解所得的优化问题, 我们使用割平面算法 (cutting plane algorithm) 来求解。为了提高算法的效率, 解决约束条件过多的问题, 在算法的每一步我们都只考虑那些最重要的约束条件。从直观上讲, 要得到好的 \mathbf{w} , 我们需要集中在那些使得 ε_i 较大的约束条件, 并在求解时重点考虑这些约束条件以加快计算。我们使用一个重点错误排序的集合 \mathcal{W} 来简化计算: 对于每个查询 q_i , 我们用 \mathcal{W}_i 来记录当前查询 q_i 应考虑的重点错误排序的集合, 而 \mathcal{W}_i 中只包含当前使得对应的 ε_i 较大的重点错误排序。 \mathcal{W} 表示所有应考虑的重点错误排序的集合, 即 $\mathcal{W} = \cup_i \mathcal{W}_i$ 。在算法开始时, \mathcal{W} 是空集; 在算法的每个循环中, 我们根据当前的 \mathbf{w} 来更新 \mathcal{W}

以及相应的 \mathcal{W} ，直到算法收敛到一个局部最优解。具体来说，对于查询 q_i 的某个错误的排序结果 π ，我们可以定义 π 导致的约束条件的违反程度如下：

$$H(\pi; \mathbf{w}) = 1 - AP(\pi) + F(x_i, \pi; \mathbf{w}) - F(x_i, \pi_i; \mathbf{w}) \quad (8-60)$$

$H(\pi; \mathbf{w})$ 越大，说明导致的 ε_i 越大，对应约束条件的优先级越高。算法的具体步骤见算法 8-2。

算法 8-2 SVM^{map} 算法

```

输入：训练集  $(x_1, \pi_1), \dots, (x_m, \pi_m)$ ，控制参数  $C, t$ 
对所有的  $i=1, 2, \dots, m$ ，设置  $\mathcal{W}_i = \emptyset$ 
repeat
  for  $i=1, 2, \dots, m$  do
    计算  $\hat{\pi} = \arg \max_{\pi \in \Pi_i} H(\pi; \mathbf{w}) = 1 - AP(\pi) + F(x_i, \pi; \mathbf{w}) - F(x_i, \pi_i; \mathbf{w})$ 
    计算  $\varepsilon_i = \max \left\{ 0, \max_{\pi \in \mathcal{W}_i} H(\pi; \mathbf{w}) \right\}$ 
    if  $H(\hat{\pi}; \mathbf{w}) > \varepsilon_i + t$  then
       $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{\hat{\pi}\}$ 
       $\mathcal{W} = \bigcup_i \mathcal{W}_i$ 
    使用当前的  $\mathcal{W}$ ，利用割平面算法求解式 (8-57) 中的优化问题
  endif
endfor
until 所有的  $\mathcal{W}_i$  都不变
  
```

根据以上算法的具体步骤，在每次循环中，我们要对每个查询 q_i 计算 $\hat{\pi} = \arg \max_{\pi \in \Pi_i} H(\pi; \mathbf{w})$ 。如

果我们遍历所有可能的 π 的话，算法的效率会受到影响。这里我们利用 MAP 的具体性质以加快计算。具体而言，就是对于查询 q_i 对应的文档的排序结果，我们并不需要关心每个位置对应哪个文档，而只需要关心该位置对应的文档是否相关即可。对于一个排序结果，即使我们交换两个相关文档的位置，也并不会改变 AP 的值。在 SVM^{map} 算法中，利用该性质，将考虑查询 q_i 对应的约束条件的复杂度降为 $O(N_i \log N_i)$ 。由于本节着重说明算法的原理而非具体实现，因此我们就省略加快计算的具体细节了。这里参数 $t > 0$ 是我们用来判断是否将 $\hat{\pi}$ 加入 \mathcal{W}_i 的容许参数。

注意，在 $\Psi(x_i, \pi)$ 的定义中，我们实际上考虑了 $d_{i,1}, d_{i,2}, \dots, d_{i,N_i}$ 中所有(相关，不相关)文档对之间的关系，因此 SVM^{map} 算法与逐对方法也有相似之处。但是 SVM^{map} 算法在优化问题中进一步考虑了所有不同的排序结果，在这个意义上该算法是一种典型的逐列方法。

SVM^{map} 算法目前已经成功地推广到其他算法评价标准，包括 NDCG 和 MRR。其基本思想与 SVM^{map} 算法相同，核心就是在约束条件中引入 NDCG 和 MRR，并利用 NDCG 和 MRR

的性质以快速找出使得 ε_i 最大的约束条件。由于篇幅所限，本节的一些内容我们就不展开讨论了。

8.5.2 讨论

与逐点方法和逐对方法相比，逐列方法直接考虑了排序算法的整个输出，并优化对应的评价标准（如 MAP 等）。逐列方法更直接地考虑了影响排序算法性能的因素。例如，排序结果中的位置信息被明确考虑了，同时也考虑了每个查询对应的文档之间的相互关系。从理论上讲，逐列方法应该更易于取得较高的性能。但是另一方面，在实际使用中要注意逐列方法的计算复杂度较高。

第9章 集成学习

9.1 集成学习简介

作为本书的最后一章，我们主要介绍集成学习。首先我们介绍集成学习的基本思想，之后讨论几种常用的集成学习的方法，包括 bagging、boosting 和 stacking。对于 bagging 和 boosting，我们进一步介绍几种在实际中非常有效的算法，包括随机森林、AdaBoost 及提升决策树。

集成学习的核心是构建多个不同的模型，并将这些模型聚合起来从而提高模型的性能。集成学习适用于机器学习的几乎所有领域，包括我们前面讨论过的回归、分类、推荐和排序等。在集成学习中，构建的一系列模型称为基学习器。通过使用不同的策略，我们可将这些基学习器聚合起来。在集成学习中，通常并不要求每个基学习器的性能特别好。在很多典型的集成学习算法中，我们只要求每个基学习器要好于随机猜测（random guess）。例如，在两类分类问题中，我们要求基学习器的准确率大于 0.5 即可。一般而言，这种基学习器也称为弱学习器（weak learner）。在集成学习中，重点是使训练得到的基学习器满足多样性（diversity）的要求，这样将多个基学习器聚合起来时，我们能够有效地提高性能。从直观上看，虽然每个基学习器都犯错，但是如果它们在犯不同的错误，那么将它们聚合起来之后犯错的可能性会很低。反之，如果基学习器比较相似，则通过集成学习提高性能的幅度较小，甚至可能带来过拟合（over-fitting）的问题而导致性能降低。一个极端的例子是，如果我们得到多个完全一样的模型，那么将它们聚合起来不会得到任何提升。

由于集成学习有效地考虑了多个不同的模型，一般而言能够获得较好的性能，因此在很多注重算法性能的场合，集成学习一般是首选。例如，在很多数据挖掘的竞赛中，获胜的算法一般都是使用集成学习将多个模型聚合而成。

根据基学习器的生成策略，集成学习的方法可以分为两类：（1）并行方法（parallel method），以 bagging 为主要代表；（2）顺序方法（sequential method），以 boosting 为主要代表。

在并行方法中，我们同时构建多个基学习器，并利用这些基学习器的独立性以提高最后模型的性能。在集成学习中，我们要尽量构建独立的学习器，以使得它们犯错也相互独立。当然，在实际中我们很难得到完全相互独立的基学习器。此外，我们一般只有一个训练集。如果都从同一训练集得到多个不同的模型的话，它们之间很难做到相互独立。因此，在实践中，我们通常通过引入随机性来尽量构造相互独立的模型。并行方法的一大优点是利于并行计算，这样我们可以显著地降低训练模型所需要的时间。

在顺序方法中，我们顺次构建多个学习器。当我们构建后面的学习器时，希望后面的学

习器能够避免前面学习器的错误,从而提高聚合后的性能。因此,在顺序方法中,我们要利用基学习器的相关性。与并行方法不同,在顺序方法中,我们需要逐个建立新的学习器,较难利用并行性以缩短训练时间。

与单个模型相比,集成学习的缺点包括:(1)计算复杂度较大。因为在集成学习中需要训练多个模型,所以计算复杂度会有较大程度的提高;(2)一般而言,所得的模型很难解释。如单个决策树模型很容易解释,而由多个决策树组成的随机森林(random forest)却不大容易解释。

在本章中,我们首先介绍顺序方法中的 bagging,并介绍 bagging 的一个典型应用:随机森林;然后我们介绍顺序方法中的 boosting,并介绍 boosting 的两个典型应用:AdaBoost 和梯度提升算法;最后,我们介绍 stacking 的基本思想及应用。

在下面的讨论中,我们一般以分类问题为例进行讨论。本章所讨论的集成学习方法都可以简单修改使之能适用于更广泛的问题,如回归问题、排序和推荐问题等。这也是我们将集成学习放到最后一章讨论的原因之一。

本章中我们将训练集表示为 $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, 这里 $x_i \in \mathbb{R}^d$ 、 $y_i \in \{-1, 1\}$ 。在集成学习中,我们通常构建 m 个弱学习器,记为 $h_j(x)$, $j = 1, 2, \dots, m$ 。一般而言,我们通常考虑多个弱学习器的线性组合作为最终的模型。在聚合 m 个弱学习器时,将 $h_j(x)$ 的权重记为 w_j 。这样,最终的模型记为:

$$f(x) = \sum_{j=1}^m w_j h_j(x) \quad (9-1)$$

在分类问题中,假设输出是 $\{-1, 1\}$, 则可将最终的模型记为:

$$f(x) = \text{sign} \left(\sum_{j=1}^m w_j h_j(x) \right) \quad (9-2)$$

这里 sign 是符号函数。在集成学习中,一般而言,我们需要从训练集中学习出 m 个学习器 $h_j(x)$, $j = 1, 2, \dots, m$, 同时也需要学习 w_j , $j = 1, 2, \dots, m$ 。

9.2 bagging 简介

在本节中,我们讨论并行集成学习算法的一个典型例子——bagging。在下一节中我们介绍最常用的 bagging 算法随机森林。

bagging 是 Bootstrap AGGREGatING 的缩写。简而言之,就是通过 bootstrap 取样(可重复取样)的方法构造多个不同的训练集。之后在每个训练集上训练相应的基学习器,最后将这些基学习器聚合起来得到最终的模型。从定义来看, bagging 中有两个重要的部分:(1)bootstrap 取样;(2)模型聚合(aggregation)。下面我们分别予以介绍。

首先我们讨论 bootstrap 取样。在前面的讨论中，我们强调了基学习器相互独立的重要性：虽然每个基学习器的性能较弱，但是如果它们都是从不同的角度犯错，那么将它们聚合起来的性能也许会有较大提升。在实际中，在给定一个训练集的情况下，如何尽可能地建立相对独立的基学习器呢？基本上我们只有两种选择：（1）在训练每个基学习器的时候使用不同的训练集；（2）在训练集相同或者相近的情况下，使用不同的学习算法来训练不同的基学习器。如果我们选择使用不同的训练集，一种可能性是将整个训练集划分为多个不同的子集，然后在每个子集上训练不同的基学习器。这个方法虽然保证了这些基学习器是从不同的训练集上得到的，但是由于每个子集都显著小于原来的训练集，使得构建的基学习器可能遗漏了原始训练集中的一些关键信息。同时，每个子集和整个训练集的分布可能存在差异。因此，这样得到的基学习器的性能会受到影响。

因此，这里的问题是，既要利用训练集中更多的样本，又要同时尽量保证不同的训练集的独立性。为了解决这个问题，人们提出了 bootstrap 取样（bootstrap sampling）。简而言之，bootstrap 取样就是使用可重复取样（sampling with replacement）的方法，从样本数为 n 的数据集中取出 n 个样本。除了一种极端情况（就是 bootstrap 取样得到了整个原来的数据集），使用 bootstrap 取样得到的 n 个样本中都有重复样本，同时也有原来数据集中的样本没有被取到。注意，在可重复取样中，我们假设每个样本被选中的概率是一样的。在 bagging 中，对于原始的训练集，我们使用 bootstrap 取样 m 次，选取出 m 个样本集。在每个样本集上，我们构建相应的基学习器。这样就得到了 m 个不同的学习器。

bagging 的另一个重要步骤是模型聚合。在 bagging 中，我们通常采用比较简单的方法来聚合多个模型。对于分类问题，我们采用投票（voting）的方法将 m 个基学习器的分类结果中出现最多的一个作为最终的分类结果；对于回归问题，我们直接取 m 个基学习器的输出的平均值。注意，使用 bagging 还可以处理多类分类问题，只需要基学习器能够处理多类问题即可。

算法 9-1 给出了 bagging 在分类和回归问题中的基本流程。其中 $mode$ 函数计算统计中的众数（mode）， $mode(f_1(x), \dots, f_m(x))$ 表示从 $f_1(x), \dots, f_m(x)$ 中选取出现频率最高的类标作为最终的分类结果。

算法 9-1 bagging 的基本流程

1. for $j=1:m$
 - 1.1 产生一个 bootstrap 取样的样本集 S_j
 - 1.2 在 S_j 上训练一个基学习器 $f_j(x)$

2. 聚合 m 个模型：

2.1 对于分类问题：

$$f(x) = mode(f_1(x), \dots, f_m(x)) \quad (9-3)$$

2.2 对于回归问题：

$$f(x) = \frac{1}{m} \sum_{j=1}^m f_j(x) \quad (9-4)$$

bagging 的另一个优点：对于每个基学习器 $f_j(x)$ ，我们可以利用不在训练集 S_j 中的样本 (out-of-bag sample, OOB 样本) 来估计基学习器的性能。在训练模型的过程中，由于我们从未接触过 OOB 样本，因此它们是天然的检验算法性能的数据。

这里简单介绍如何利用 OOB 样本来估计 bagging 在分类问题中的正确率。在后面讨论 bagging 的具体例子随机森林时，我们将进一步讨论如何使用 OOB 样本来估计变量的重要性。

对于每个样本 x_i ，我们首先可以找出所有没有利用 x_i 训练的基学习器 f_j ，然后利用这些基学习器计算 $f_j(x)$ ，并使用投票的方法聚合这些 $f_j(x)$ 得到最终的输出。对于所有的 n 个样本，我们都可重复这一过程。最后，将这些样本的预测值和真实类别相比较即可算出相应的正确率。

虽然一次 bootstrap 取样能够得到 n 个样本（这里 n 是原始训练集中的样本数目），但是在理想情况下，只能得到大约 63.2% 的原始样本。下面我们详细解释为什么只能得到大约 63.2% 的原始样本。这一段介绍不影响读者学习 bagging，不感兴趣的读者可以跳过。

考虑样本 $x_i (i=1, 2, \dots, n)$ ，其在 n 次取样中都没有被选中的概率 P_i^0 为：

$$P_i^0 = \left(1 - \frac{1}{n}\right)^n \quad (9-5)$$

因此， x_i 被选中至少一次的概率 P_i^1 为：

$$P_i^1 = 1 - \left(1 - \frac{1}{n}\right)^n \quad (9-6)$$

当 n 趋近于无穷大时，有：

$$\lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \quad (9-7)$$

因此

$$\lim_{n \rightarrow \infty} P_i^1 = 1 - \frac{1}{e} = 0.632 \quad (9-8)$$

因此，如果我们考虑期望值的话，当 n 很大时，就会有大约 63.2% 的原始样本在一次 bootstrap 取样中被选中。

讨论

在回归分析中，我们讨论了偏差-方差权衡 (bias-variance tradeoff)。我们知道，模型 (model) 也有偏差 (bias) 和方差 (variance)。从直观上看，对于一组给定的训练数据，如果某一算法能够很好地拟合这组训练数据，则该模型的偏差较低；但是如果训练数据发生了一些变动，而导

致算法产生的模型也发生了较大变动的话,则该模型的方差较高。典型的例子就是决策树和人工神经网络。虽然它们都能很好地拟合训练集,但是对于训练集过于敏感。下面我们首先以随机变量为例讨论多个变量平均之后的方差,然后将其推广到模型以说明模型多样性的重要性。

我们知道,假设我们有 m 个独立同分布的随机变量,它们的方差是 σ^2 , 则平均之后的方差为 $\frac{\sigma^2}{m}$ 。简而言之,取平均值可以降低方差。注意,在 bagging 中,实际上不可能构造出完全独立的基学习器。如果我们将 bagging 中构造的每个模型视为一个随机变量的话,则这些随机变量都不是相互独立的。

下面我们考虑稍微一般的情况:考虑 m 个随机变量同分布但不相互独立,并假设两两之间的相关系数为 $\rho > 0$, 则平均后的方差为:

$$\rho\sigma^2 + \frac{1-\rho}{m}\sigma^2 \quad (9-9)$$

从式 (9-9) 可以看出,当 m 增加时,式 (9-9) 中的第二项减少,但是第一项不受 m 的影响。为了清楚地了解 ρ 对平均后的方差的影响,可将式 (9-9) 写为:

$$\frac{\sigma^2}{m} + \sigma^2 \left(1 - \frac{1}{m}\right) \rho \quad (9-10)$$

从式 (9-10) 可以看出, ρ 越小,平均之后的方差就越小。回到模型的讨论,如果所构建的模型相互越独立,则平均之后的模型对应的方差就越小。

这里我们给出随机变量平均后的方差的具体推导过程。不感兴趣的读者可以直接跳过这部分的推导。假设我们有未知变量 v_j ($j=1,2,\dots,m$) 满足 $\text{var}(v_j) = \sigma^2$, 并且当 $j_1 \neq j_2$ 时,对应的相关系数 $\text{cor}(v_{j_1}, v_{j_2}) = \rho$, 则协方差 $\text{cov}(v_{j_1}, v_{j_2})$ 可写为:

$$\text{cov}(v_{j_1}, v_{j_2}) = \text{cor}(v_{j_1}, v_{j_2}) \sqrt{\text{var}(v_{j_1})} \sqrt{\text{var}(v_{j_2})} = \rho\sigma^2 \quad (9-11)$$

v_1, v_2, \dots, v_m 的平均值 $\bar{v} = \frac{1}{m}(v_1 + v_2 + \dots + v_n)$ 的方差可计算如下:

$$\begin{aligned} \text{var}(\bar{v}) &= \text{var}\left(\frac{1}{m}(v_1 + v_2 + \dots + v_n)\right) = \sum_{j=1}^m \frac{1}{m^2} \text{var}(v_j) + \sum_{j_1 \neq j_2} \frac{1}{m^2} \text{cov}(v_{j_1}, v_{j_2}) \\ &= m \frac{1}{m^2} \sigma^2 + m(m-1) \frac{1}{m^2} \rho\sigma^2 = \frac{1}{m} \sigma^2 + \frac{m-1}{m} \rho\sigma^2 = \rho\sigma^2 + \frac{1-\rho}{m} \sigma^2 \end{aligned} \quad (9-12)$$

在集成学习中,可以将模型的方差类比为随机变量的方差。通过平均多个模型,也能降低模型的方差,从而得到更好的模型。bagging 通过人为的办法,构建了多个模型,最后通过平均的方法,降低模型的方差。通过上面的讨论可知,要尽量发挥 bagging 的功效,在构造基学习器时要尽量构造相互独立的基学习器,这样最后平均得到的模型的方差就会小。另外, bagging 对于那些高方差、低偏差 (high-variance, low-bias) 的基学习器非常有效。基本上基

学习器对训练数据越敏感, bagging 之后的效果就越好。但是如果基学习器比较稳定, 换言之就是方差较小, 在这种情况下, bagging 所能取得的提升则非常有限。此外, bagging 的优点是非常适合并行训练多个算法, 可有效地处理大数据。

9.3 随机森林

在本节中, 我们讨论 bagging 的一个实际应用算法——随机森林 (random forest)。随机森林使用了 bagging 的基本思想来训练一系列决策树。但是随机森林又根据决策树的特点做了很多改进, 使得所构建的决策树尽量没有相关性, 从而显著提高最终的性能。在很多实际问题中, 随机森林都能取得很好的效果, 同时由于其控制参数易于选择, 因此使得随机森林成为实际中非常受欢迎的算法, 广泛应用于各类实际问题中。

在讨论随机森林之前, 我们先回顾一下决策树。决策树能够有效地获取多个变量之间的相互关系。同时, 如果决策树足够深, 那么它能够有效地降低模型的偏差。通过前面章节关于决策树的讨论我们知道, 较深的决策树虽然能够降低模型的偏差, 但是很容易导致过拟合。对于不同的数据集, 得到的决策树可以存在较大差异。换言之, 决策树的方差较大, 因此是使用 bagging 技巧的好对象。

9.3.1 训练随机森林的基本流程

在本节中, 我们介绍训练随机森林的基本思想和流程。随机森林的基本思想是利用 bagging 构建很多决策树。通过前一节的讨论我们知道, 为了提高 bagging 的效果, 我们需要构建尽量独立的基学习器。与简单的 bagging 相比, 随机森林在构建每棵决策树时, 通过引入一些随机信息, 有效地降低了各个基学习器的相关度。

具体来说, 在构建决策树时, 我们需要在每步选择最优的变量。在标准的构建决策树的算法中, 我们需要考虑当前所有可选的变量; 但是在随机森林中, 我们从所有可选的 d 个变量中随机地取出 d_1 个变量, 然后从中选取出最优的变量。从直观上讲, 降低 d_1 的值虽然降低了单个决策树在其对应的训练集上的表现, 但是它能够降低所构建的不同的决策树的相关度, 从而能够降低最终平均后的模型的方差。训练随机森林的具体步骤参见算法 9-2。

算法 9-2 训练随机森林的具体步骤

1. for $j=1:m$
 - 1.1 产生一个 bootstrap 取样的样本集 S_j
 - 1.2 在 S_j 上训练一个决策树 T_j 。在生成决策树的过程中, 对于每个对应的样本数大于 n_{min} 的叶结点做如下处理:
 - 1.2.1 从所有可选的 d 个变量中随机选择 d_1 个变量
 - 1.2.2 从这 d_1 个变量中选择导致最优划分的变量

1.2.3 将该结点根据选择的最优变量划分为两个子结点

1.2.4 重复该过程, 直到所有的叶结点对应的样本数都小于或等于 n_{min}

2. 聚合 m 棵决策树 T_1, T_2, \dots, T_m :

2.1 对于分类问题:

$$f(\mathbf{x}) = \text{mode}(T_1(\mathbf{x}), \dots, T_m(\mathbf{x})) \quad (9-13)$$

2.2 对于回归问题:

$$f(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^m T_j(\mathbf{x}) \quad (9-14)$$

在随机森林中, 我们要构建一系列决策树。在构建每个决策树时, 首先使用 bootstrap 取样得到一个样本数为 n 的可重复样本。然后利用这个样本集, 构建一棵决策树。在构建决策树的每一步, 我们都是按照上面所讨论的, 先随机取出 d_1 个变量, 再选取出最优的变量。最后我们将所有模型的输出取平均 (回归问题) 或者取出最多的类别 (分类问题) 作为最终的输出。

9.3.2 利用随机森林估计变量的重要性

在实际使用机器学习算法时, 通常数据中都含有噪声或者冗余数据。因此, 如果模型能够直接告诉我们各个变量的重要性, 就可以剔除这些噪声和冗余数据。这样一方面能够提高模型的性能, 另一方面能够降低计算复杂度。

在随机森林中, 有两种方法可以用来确定每个变量的重要性: 第一种方法是利用决策树的性质来计算变量的重要性; 第二种方法是利用 OOB 样本来估计变量的重要性。在实际使用中, 第二种方法的应用更为广泛。

在第一种方法中, 在构建每棵决策树时, 可以计算每个结点对应的变量所导致的信息增益。把所有的决策树都聚合起来, 就可以计算每个变量所导致的信息增益。利用这个信息增益, 可计算每个变量的重要性: 信息增益越大, 变量的重要性越高。

在第二种方法中, 可以使用 OOB 样本来确定每个变量的重要性。在考虑第 j 棵树 T_j 时, 我们将所有没有包含在第 j 次 bootstrap 取样集 S_j 中的 OOB 样本集记为 O_j 。我们可将 O_j 作为检验集 (validation set), 计算 T_j 在 O_j 上的性能 P_j , 如分类中的准确率或者回归中的 RMSE。当考虑第 k 个变量的重要性时, 我们将 O_j 中所有样本的第 k 个变量的值打乱以得到一个随机排列。这样我们得到一个新的数据集 O'_{jk} , 而且 O'_{jk} 与 O_j 的区别在于, O'_{jk} 中第 k 个变量的值是随机赋予的, 并没有明确的意义。我们将 O'_{jk} 作为一个新的检验数据集, 并计算 T_j 在 O'_{jk} 上的性能 P'_{jk} 。一般而言, P_j 要好于 P'_{jk} , 我们将 P_j 与 P'_{jk} 的差作为第 k 个变量的重要性。注意, 在随机森林中, 我们有 m 棵树, 将所有 m 棵树 P_j 与 P'_{jk} 的差平均起来, 就可以作为第 k 个变

量的重要性的度量。以分类问题为例,假设我们选定性能度量为准准确率,则一般而言有 $P_j \geq P'_{jk}$,那么第 k 个变量的重要性可以使用式 (9-15) 来计算:

$$IM(v_k) = \frac{1}{m} \sum_{j=1}^m (P_j - P'_{jk}) \quad (9-15)$$

这种通过随机排列变量值来估计变量重要性的方式,目前也有了应用。在微软公司最新推出的 Azure Machine Learning^①中就有相应的模块来通过此法估计变量的重要性^②。

9.3.3 随机森林的实际使用

在实际使用随机森林时,一般需要确定如下参数。

- 决策树的数目 m ;
- 每棵决策树的大小,由决策树叶结点所能包含的样本数的最大值决定;
- 每次选取最佳变量时随机选取的变量数 d_1 。

这里我们首先讨论选取这些参数的一般原则,然后通过介绍 R 中的 randomForest 包来具体介绍随机森林的使用。

对于 d_1 的选取和决策树的大小,随机森林的发明者 Leo Breiman 提出了如下建议。

- 在分类问题中,可以将 d_1 的默认值设为 $d_1 = \lfloor \sqrt{d} \rfloor$,同时决策树的结点至少包括 1 个样本。
- 在回归问题中,可以将 d_1 的默认值设为 $d_1 = \lfloor d/3 \rfloor$,同时决策树的结点至少包括 5 个样本。

对于 d_1 的值, Breiman 提出可以试验默认值、默认值的一半以及默认值的两倍,并从中挑选最优值。在很多数据中,随着 d_1 的变化,随机森林的性能变化并不是非常剧烈。在一些数据集中,甚至将 d_1 设为 1 都能取得良好的效果。然而,如果数据集的维数 d 较大但是其中很多是无用噪声的话,在建模中只有少数的特征是真正有用的。在这种情况下,我们需要选取较大的 d_1 值,这样那些有用的特征才能够在构建决策树时被选到。

在实际中,如果不是数据集特别小或者问题特别简单,那么上百棵决策树是必需的。一般而言,随着变量的增加,若要取得较好的性能,就需要增加随机森林中决策树的数目。具体来说,我们可以简单测试当前决策树的数目是否足够:比较当前已经得到的所有决策树及其一个部分。比如,目前有 500 棵决策树,我们可以选取前面的 450 棵决策树,比较 500 棵决策树组成的随机森林模型和 450 棵决策树组成的随机森林模型。换言之,我们利用两个不同的随机森林模型来看它们的预测值是否有较大的区别。如果区别较大,则意味着我们要继续增加决策树的数目。我们的目标是保证决策树的数目足够多,直到随机森林的性能比较稳定为止。

R 程序以及介绍

下面介绍 R 中非常流行的软件包 randomForest^③。我们提供了程序文件 randomForest_classification_example.R 和 randomForest_regression_example.R 用来介绍如何应用随机森林解

① <https://studio.azureml.net/>

② 该模块是 Permutation Feature Importance 模块。

③ <https://cran.r-project.org/web/packages/randomForest/index.html>

决分类问题和回归问题。randomForest 包中的核心函数是 randomForest 函数。使用该函数，可以构建一个随机森林模型。该函数的重要参数包括以下几个。

- data: 训练数据。
- 公式: 用来指定哪一列是目标变量，哪些列是自变量。
- ntree: 随机森林中决策树的数目。
- mtry: 每次选取最佳变量时随机选取的变量数 d_1 。
- importance: 取值为 TRUE 或者 FALSE，表示是否计算变量的重要性。

其他一些函数如下。

- combine: 合并多个随机森林模型得到一个最终的随机森林模型。
- grow: 对于当前的随机森林模型，继续生成新的决策树。
- importance: 显示变量的重要性（文本形式）。
- varImpPlot: 作图显示变量的重要性（图像形式）。

注意，randomForest 包的 randomForest 函数中有一个参数是 importance，同时 randomForest 包中也有一个函数为 importance。

在下面的例子中，我们着重讲解 randomForest_regression_example.R，以说明如何使用 randomForest 包。在该文件中，我们利用 randomForest 包，完成以下任务：

- 导入数据和进行简单的数据分析。
- 将整个数据集分为训练集和测试集。
- 作为参照，训练一个决策树模型。
- 使用不同的参数训练两个不同的随机森林模型。
- 首先生成 3 个不同的随机森林模型，然后将它们合并成为一个随机森林模型。
- 首先生成一个随机森林模型，然后不断增加其中决策树的数目。
- 使用随机森林研究变量的重要性。

下面是具体的 R 代码。注意，我们首先要检查 randomForest 包有没有安装，如果没有安装，则首先安装该包。

```
# Check required package is installed or not. If not, install it.
randomForest.installed <- 'randomForest' %in% rownames(installed.packages())
if (randomForest.installed) {
  print("the randomForest package is already installed, let's load it...")
}else {
  print("let's install the randomForest package first...")
  install.packages('randomForest', dependencies=T)
}
library('randomForest')

#=====
# Step 1. Load the data and simple exploration
# load the mtcars data
data(mtcars)
```

```

D <- mtcars

# show the type for each col
for(i in 1:ncol(D)) {
  msg <- paste('col ', i, ' and its type is ', class(D[,i]))
  print(msg)
}

# Step 2. Split the data into training and test sets
# Randomly split the whole data set into a training and a test data set
# After splitting, we have the training set: (X_train, y_train)
# and the test data set: (X_test, y_test)
train_ratio <- 0.7
n_total <- nrow(D)
n_train <- round(train_ratio * n_total)
n_test <- n_total - n_train
set.seed(42)
list_train <- sample(n_total, n_train)
D_train <- D[list_train,]
D_test <- D[-list_train,]

y_train <- D_train$mpg
y_test <- D_test$mpg

# Step 3. Benchmark: train a single decision tree using rpart
library('rpart')
M_rpart1 <- rpart(mpg~., data = D_train)
print('show the summary of the trained model')
summary(M_rpart1)

# Compute the performance on the training and test data sets
y_test_pred_rpart1 <- predict(M_rpart1, D_test)
rmse_test_rpart1 <- sqrt(sum((y_test - y_test_pred_rpart1)^2) / n_test)
msg <- paste0('rmse_test_rpart1 = ', rmse_test_rpart1)
print(msg)

y_train_pred_rpart1 <- predict(M_rpart1, D_train)
rmse_train_rpart1 <- sqrt(sum((y_train - y_train_pred_rpart1)^2) / n_train)
msg <- paste0('rmse_train_rpart1 = ', rmse_train_rpart1)
print(msg)

# Step 4. train 2 random forest models using different parameters
# Step 4.1 Train a random forest model using default parameters
# the default number of trees is 500
M_randomForest1 <- randomForest(mpg~., data = D_train)
print('show the summary of the trained model')
summary(M_randomForest1)

# We can get mean of squared residuals using the print function directly

```

```

print(M_randomForest1)
# We can check the number of trees of the trained model
ntree1 <- M_randomForest1$ntree
msg <- paste0('number of trees in M_randomForest1 = ', ntree1)
print(msg)

# Get the prediction on the test set and compute the RMSE
y_test_pred_rf1 <- predict(M_randomForest1, D_test)
rmse_test_rf1 <- sqrt(sum((y_test - y_test_pred_rf1)^2) / n_test)
msg <- paste0('rmse_test_rf1 = ', rmse_test_rf1)
print(msg)

y_train_pred_rf1 <- predict(M_randomForest1, D_train)
rmse_train_rf1 <- sqrt(sum((y_train - y_train_pred_rf1)^2) / n_train)
msg <- paste0('rmse_train_rf1 = ', rmse_train_rf1)
print(msg)

# We train a second model using fewer decision trees, and control the
# complexity of each tree.
M_randomForest2 <- randomForest(mpg~., data = D_train, ntree=50, mtry=3)
print('show the summary of the trained model')
summary(M_randomForest2)
print(M_randomForest2)
# We can check the number of trees of the trained model
ntree2 <- M_randomForest2$ntree
msg <- paste0('number of trees in M_randomForest2 = ', ntree2)
print(msg)

# Get the prediction on the test set and compute the RMSE
y_test_pred_rf2 <- predict(M_randomForest2, D_test)
rmse_test_rf2 <- sqrt(sum((y_test - y_test_pred_rf2)^2) / n_test)
msg <- paste0('rmse_test_rf2 = ', rmse_test_rf2)
print(msg)

y_train_pred_rf2 <- predict(M_randomForest2, D_train)
rmse_train_rf2 <- sqrt(sum((y_train - y_train_pred_rf2)^2) / n_train)
msg <- paste0('rmse_train_rf2 = ', rmse_train_rf2)
print(msg)

# Step 5. Combine 3 random forest models together using the combine function
# Train 3 random forest models
M_rf_base1 <- randomForest(mpg~., data = D_train, ntree = 15)
M_rf_base2 <- randomForest(mpg~., data = D_train, ntree = 20)
M_rf_base3 <- randomForest(mpg~., data = D_train, ntree = 10)
# Combine these 3 models just trained
M_rf_comb <- combine(M_rf_base1, M_rf_base2, M_rf_base3)
print(M_rf_comb)

```

```

# compute the performance on the test data set
y_test_pred_rf_base1 <- predict(M_rf_base1, D_test)
rmse_test_rf_base1 <- sqrt(sum((y_test - y_test_pred_rf_base1)^2) / n_test)
msg <- paste0('rmse_test_rf_base1 = ', rmse_test_rf_base1)
print(msg)

y_test_pred_rf_base2 <- predict(M_rf_base2, D_test)
rmse_test_rf_base2 <- sqrt(sum((y_test - y_test_pred_rf_base2)^2) / n_test)
msg <- paste0('rmse_test_rf_base2 = ', rmse_test_rf_base2)
print(msg)

y_test_pred_rf_base3 <- predict(M_rf_base3, D_test)
rmse_test_rf_base3 <- sqrt(sum((y_test - y_test_pred_rf_base3)^2) / n_test)
msg <- paste0('rmse_test_rf_base3 = ', rmse_test_rf_base3)
print(msg)

y_test_pred_rf_comb <- predict(M_rf_comb, D_test)
rmse_test_rf_comb <- sqrt(sum((y_test - y_test_pred_rf_comb)^2) / n_test)
msg <- paste0('rmse_test_rf_comb = ', rmse_test_rf_comb)
print(msg)

# Step 6. Continuously grow a random forest by training more trees, and
# plot the RMSE of training and test as the number of trees increases.

ntree_list <- 1:200
ntree_length <- length(ntree_list)
rmse_train_list <- rep(0, ntree_length)
rmse_test_list <- rep(0, ntree_length)
for (i in 1:ntree_length) {
  # Build the random forest model based on the existing random forest model
  if (i==1) {
    M_rf_base <- randomForest(mpg~., data = D_train, ntree = ntree_list[1])
  } else {
    ntree_delta <- ntree_list[i] - ntree_list[i-1]
    M_rf_base <- grow(M_rf_base, ntree_delta)
  }

  # Compute rmse on training and test data set
  y_train_pred_rfi <- predict(M_rf_base, D_train)
  y_test_pred_rfi <- predict(M_rf_base, D_test)
  rmse_train_rfi <- sqrt(sum((y_train - y_train_pred_rfi)^2) / n_train)
  rmse_test_rfi <- sqrt(sum((y_test - y_test_pred_rfi)^2) / n_test)
  rmse_train_list[i] <- rmse_train_rfi
  rmse_test_list[i] <- rmse_test_rfi
}

# Plot the training and test RMSE
y_min <- min(min(rmse_test_list), min(rmse_train_list)) - 0.1
y_max <- max(max(rmse_test_list), max(rmse_train_list)) + 0.1
plot(range(ntree_list), c(y_min, y_max), type='n', xlab='ntree', ylab='RMSE')

```



```

lines(ntree_list, rmse_train_list, type='l', lty=1, col='black')
lines(ntree_list, rmse_test_list, type='l', lty=2, col='red')
legend_char_list <- c('training data', 'test data')
# We can use locator(1) to specify the legend position by clicking the mouse
# legend(locator(1), legend_char_list, cex=1.2, col=c('red', 'black'), lty=c(1,2))
# Or we can specify the legend position directly
legend("topright", legend_char_list, cex=1.2, col=c('red', 'black'), lty=c(1,2))

# Step 7. Check the importance of all variables
# We train a new random forest model consisting of 100 trees
M_rf_imp <- randomForest(mpg~., data = D_train, ntree = 100, importance = T)
print('we show the variable importance using OOB samples')
var_imp1 <- importance(M_rf_imp, type=1)
print(var_imp1)

print('we show the variable importance using tree node impurity/MSE decrease')
var_imp2 <- importance(M_rf_imp, type=2)
print(var_imp2)

# Plot the importance of all variables
varImpPlot(M_rf_imp, main = 'Variable importance of M_rf_imp')

```

然后对数据做了一个简单的检查，即检查每列的类型，其输出如下：

```

[1] "col 1 and its type is numeric"
[1] "col 2 and its type is numeric"
[1] "col 3 and its type is numeric"
[1] "col 4 and its type is numeric"
[1] "col 5 and its type is numeric"
[1] "col 6 and its type is numeric"
[1] "col 7 and its type is numeric"
[1] "col 8 and its type is numeric"
[1] "col 9 and its type is numeric"
[1] "col 10 and its type is numeric"
[1] "col 11 and its type is numeric"

```

我们将数据按照 70%和 30%的比例分为训练集和测试集，并分别保存在数据框 `D_train` 和 `D_test` 中。

接着我们使用 `rpart` 软件包构建了一个决策树的回归模型，其输出如下：

```

[1] "show the summary of the trained model"
Call:
rpart(formula = mpg ~ ., data = D_train)
n = 22

      CP nsplit rel error   xerror   xstd
1 0.6332989    0 1.0000000 1.1516657 0.3243608
2 0.0100000    1 0.3667011 0.7476655 0.1917382

```

```

Variable importance
  hp   wt   cyl disp drat  qsec
  20   18   16   16   14   14

Node number 1: 22 observations,      complexity param=0.6332989
  mean=19.97273, MSE=40.49198
  left son=2 (12 obs) right son=3 (10 obs)
  Primary splits:
    hp < 116.5 to the right, improve=0.6332989, (0 missing)
    wt < 3.325 to the right, improve=0.6219046, (0 missing)
    cyl < 5      to the right, improve=0.6198157, (0 missing)
    disp < 163.8 to the right, improve=0.5950779, (0 missing)
    drat < 3.91  to the left,  improve=0.4739783, (0 missing)
  Surrogate splits:
    wt < 3.325 to the right, agree=0.955, adj=0.9, (0 split)
    cyl < 5      to the right, agree=0.909, adj=0.8, (0 split)
    disp < 163.8 to the right, agree=0.909, adj=0.8, (0 split)
    drat < 3.655 to the left,  agree=0.864, adj=0.7, (0 split)
    qsec < 18.25 to the left,  agree=0.864, adj=0.7, (0 split)

Node number 2: 12 observations
  mean=15.35, MSE=8.8225

Node number 3: 10 observations
  mean=25.52, MSE=22.0796

[1] "rmse_test_rpart1 = 3.60306119848109"
[1] "rmse_train_rpart1 = 3.85336924592681"

```

可以看出, 所得的决策树模型在测试集和训练集上的 RMSE 分别约为 3.60 和 3.85。之后在第 4 步中我们构建决策树模型。在第一个随机森林模型中, 全部采用默认的参数值:

```
M_randomForest1 <- randomForest(mpg~., data = D_train)
```

随后我们使用 `M_randomForest1$ntree` 检查一共构建了多少决策树。在第二个随机森林模型中, 我们指定构建 50 棵决策树, 并将 `mtry` 设为 3:

```
M_randomForest2 <- randomForest(mpg~., data = D_train, ntree=50, mtry=3)
```

我们使用 `print` 函数输出了这两个模型的一些信息:

```

[1] "show the summary of the trained model"

Call:
  randomForest(formula = mpg ~ ., data = D_train)
  Type of random forest: regression
    Number of trees: 500
  No. of variables tried at each split: 3

  Mean of squared residuals: 6.648723

```

```

% Var explained: 83.58
[1] "number of trees in M_randomForest1 = 500"
[1] "rmse_test_rfl = 1.94172203933111"
[1] "rmse_train_rfl = 1.29780945437018"
[1] "show the summary of the trained model"
Call:
  randomForest(formula = mpg ~ ., data = D_train, ntree = 50, mtry = 3)
  Type of random forest: regression
    Number of trees: 50
No. of variables tried at each split: 3
  Mean of squared residuals: 8.865426
    % Var explained: 78.11
[1] "number of trees in M_randomForest2 = 50"
[1] "rmse_test_rf2 = 1.86589255022898"
[1] "rmse_train_rf2 = 1.47045984929335"

```

在第5步中，我们构建了 `M_rf_base1`、`M_rf_base2` 和 `M_rf_base3` 随机森林模型，然后调用 `combine` 函数将这3个模型聚合起来得到一个随机森林模型。

```
M_rf_comb <- combine(M_rf_base1, M_rf_base2, M_rf_base3)
```

在第6步中，我们首先构建只有一棵决策树的随机森林模型，然后使用 `grow` 函数向该随机森林模型中不断添加新的决策树。

```
M_rf_base <- grow(M_rf_base, ntree_delta)
```

图9-1显示了随着决策树数目的增加，训练集和测试集上的RMSE的变化。从图9-1中可以看出，当决策树刚开始增加时，训练集和测试集上的RMSE都开始减小。但是当决策树的数目继续增加时，训练集和测试集上的RMSE都在波动一下后保持平稳，说明进一步增加决策树的数量不会进一步提高模型的性能。

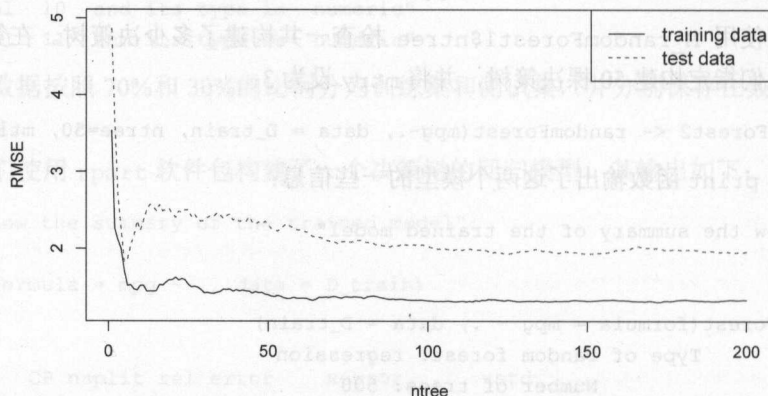


图9-1 在随机森林模型中增加决策树的数目导致的训练集和测试集上RMSE的变化情况

在第 7 步中我们考虑了随机森林模型中变量的重要程度。在构建随机森林模型时，首先必须将参数 `importance` 设为 `TRUE`。

```
M_rf_imp <- randomForest(mpg~., data = D_train, ntree = 100, importance = T)
```

然后可以使用 `importance` 函数得到该模型中变量的重要程度。通过将参数 `type` 设为 1（使用 OOB 样本来估计）或者 2（使用决策树中 MSE 或者不纯洁程度来度量），我们能够得到不同的重要程度度量。

```
var_imp1 <- importance(M_rf_imp, type=1)
var_imp2 <- importance(M_rf_imp, type=2)
```

其输出如下：

```
[1] "we show the variable importance using OOB samples"
```

```
%IncMSE
```

```
cyl 4.596421
```

```
disp 4.250417
```

```
hp 6.674625
```

```
drat 1.616390
```

```
wt 6.409588
```

```
qsec 2.719904
```

```
vs 1.652735
```

```
am 1.483702
```

```
gear 1.306683
```

```
carb 1.862388
```

```
[1] "we show the variable importance using tree node impurity/MSE decrease"
```

```
IncNodePurity
```

```
cyl 96.819302
```

```
disp 160.206895
```

```
hp 221.200911
```

```
drat 59.407580
```

```
wt 196.157244
```

```
qsec 21.761652
```

```
vs 17.399410
```

```
am 9.937528
```

```
gear 7.189293
```

```
carb 21.664176
```

最后我们使用函数 `varImpPlot` 直接绘出变量的重要程度，如图 9-2 所示。

```
varImpPlot(M_rf_imp, main = 'Variable importance of M_rf_imp')
```

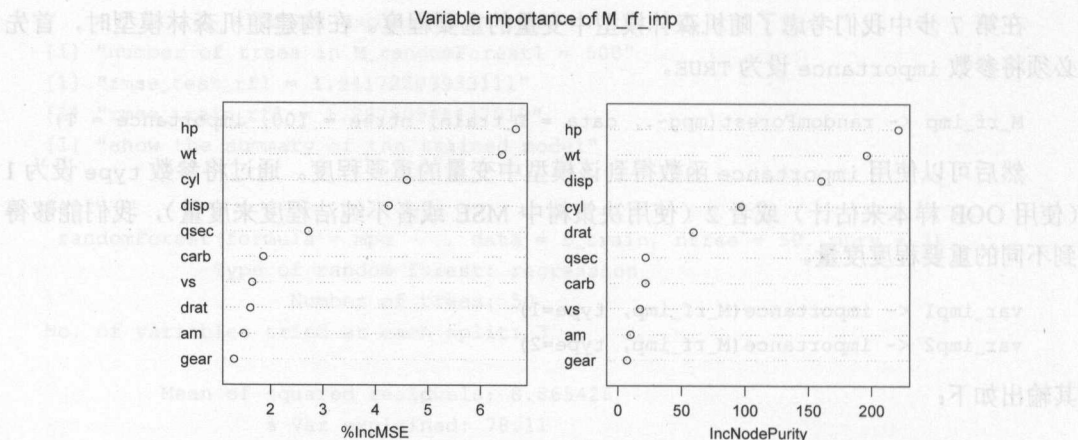



图 9-2 随机森林模型 M_rf_imp 中使用两种不同的方法所得的变量的重要程度

9.4 boosting 简介

boosting 是近年来机器学习领域比较热门的一个方向。boosting 的本意是“提高”。与 bagging 中并行构建很多基学习器相比，boosting 顺次建立一系列基学习器。在构建新的基学习器时，boosting 通过分析当前已经建立的基学习器，如分析以前的基学习器错误分类的训练样本，寻找改进的方向，来构建新的基学习器。利用 boosting，可以显著地提高传统算法的性能。

boosting 的基本思想虽然简单，但是事实上涉及的数学基础较多。在本章中，我们首先以 AdaBoost 为例来介绍 boosting。在下一节我们将介绍广泛应用的提升决策树 (boosted tree) 和对应的梯度提升 (gradient boosting) 算法。

在很多文献中，AdaBoost 都是作为 boosting 的一个典型例子来介绍的。在下面的讨论中，我们以分类问题为例介绍 AdaBoost 的基本思想，其在回归问题中的使用可以类似地推导出。

AdaBoost 的基本思想：首先构造一个弱分类器 (weak classifier)。根据该分类器在训练集上的分类结果，我们给训练集中的每个样本赋予一个权值 (weight)：如果该样本被正确分类，则权值较小；如果该样本被错误分类，则权值很大。之后我们考虑每个样本的权值以构建第二个弱分类器，使得权值较大的样本能够尽量被正确分类。换言之，在构建新的分类器时，我们尽量多考虑那些在前面被错分的样本。根据前面两个弱分类器的结果，我们可以更新每个样本的权值，从而构建第三个弱分类器。通过重复该过程，我们可以构建多个弱分类器，从而最终得到较好的分类效果。从直观上讲，后面的弱分类器集中处理前面被错分的样本，这样使得分类器犯的错误各不相同，因此聚合之后能够得到较好的效果。

在介绍 AdaBoost 算法之前，我们先介绍 boosting 的基本思想和指数损失函数 (exponential loss function)。在此基础上，我们给出 AdaBoost 的具体步骤，最后介绍如何在 R 中使用 AdaBoost。

9.4.1 boosting 和指数损失函数

在 boosting 中, 我们所得的模型 $f(\mathbf{x})$ 是顺次构建的多个模型的聚合:

$$f(\mathbf{x}) = \sum_{j=1}^m w_j h_j(\mathbf{x}; \gamma_j) \quad (9-16)$$

在式 (9-16) 中, 我们有 m 个基学习器 $h_j(\mathbf{x}; \gamma_j)$, 这里 \mathbf{x} 表示输入, γ_j 表示第 j 个模型对应的参数, w_j 是模型 $h_j(\mathbf{x}; \gamma_j)$ 对应的权重。注意, 在第 $J(J \leq m)$ 步时, 我们所得的模型 $f_J(\mathbf{x})$ 为:

$$f_J(\mathbf{x}) = \sum_{j=1}^J w_j h_j(\mathbf{x}; \gamma_j) \quad (9-17)$$

在 boosting 中, 我们顺次构建这些模型 $h_j(\mathbf{x}; \gamma_j)$, 同时 $h_j(\mathbf{x}; \gamma_j)$ 一般是一类函数, 且比较简单。例如, 在我们接下来要讨论的提升决策树中, 每个 $h_j(\mathbf{x}; \gamma_j)$ 都是由一棵决策树来确定的。

在机器学习中, 我们通常最小化模型 $f(\mathbf{x})$ 在训练集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ 上的损失函数来学习出模型的参数, 这里 $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$ 。换言之, 我们要求解如下的优化问题:

$$\min_{\{w_j, \gamma_j\}} \sum_{i=1}^n L(y_i, f(\mathbf{x}_i)) = \sum_{i=1}^n L\left(y_i, \sum_{j=1}^m w_j h_j(\mathbf{x}; \gamma_j)\right) \quad (9-18)$$

这里 L 是一个损失函数, 用来衡量 $f(\mathbf{x}_i)$ 拟合 y_i 的好坏程度。在这个优化问题中, 我们要同时求解所有的 w_j 和 γ_j 。从求解优化问题的角度讲, 同时求解所有的参数难度太大, 因此我们可以采用顺序求解的方法来求解这个优化问题。

- (1) 求解第 1 个学习器的权重 w_1 和参数 γ_1 。
- (2) 固定权重 w_1 和参数 γ_1 , 求解第 2 个学习器的权重 w_2 和参数 γ_2 。
- (3) 重复该过程, 直到求解第 m 个学习器的权重 w_m 和参数 γ_m 。

严格地讲, 在第 j 步我们求解如下最优化问题:

$$(w_j, \gamma_j) = \arg \min_{w, \gamma} \sum_{i=1}^n L(y_i, f_{j-1}(\mathbf{x}_i) + w h_j(\mathbf{x}_i; \gamma)), \quad f_{j-1}(\mathbf{x}_i) = \sum_{k=1}^{j-1} w_k h_k(\mathbf{x}_i; \gamma_k) \quad (9-19)$$

作为第 j 个弱学习器的权值和参数。换言之, 在第 j 步时, 固定住已求解出的 w_1, w_2, \dots, w_{j-1} 和 $\gamma_1, \gamma_2, \dots, \gamma_{j-1}$, 然后优化 w_j 和 γ_j 。简而言之, 就是由于同时求解所有的 w_1, w_2, \dots, w_m 和 $\gamma_1, \gamma_2, \dots, \gamma_m$ 难度太大, 因此我们采用了类似贪婪算法的方法来顺次求解该优化问题。

接下来我们讨论指数损失函数, 并以此函数引入 AdaBoost。事实上, AdaBoost 先被提出。通过分析 AdaBoost 算法, 人们发现其对应于指数损失函数。但是在本书中为了方便读者理解

AdaBoost, 我们先从指数损失函数开始讨论, 进而推导出 AdaBoost 算法。

指数函数适用于分类问题。在本章中我们假设类标 $y_i \in \{-1, 1\}$, 则指数损失函数的定义为:

$$L_{\exp}(y, f(x)) = \exp(-yf(x)) \quad (9-20)$$

图 9-3 给出了指数函数的图像, 同时也给出了其他一些常用损失函数的图像。

下面我们解释为何指数损失函数的定义是合理的。由于我们假设 $y_i \in \{-1, 1\}$, 因此, 当 $y_i = 1$ 时, 我们希望 $f(x_i)$ 越大越好; 当 $y_i = -1$ 时, 我们希望 $f(x_i)$ 越小越好, 最好是负数。综合起来, 就是希望 $y_i f(x_i)$ 越大越好。与在 SVM 中的讨论类似, 当 $y_i f(x_i) \geq 0$ 时, 分类是正确的, 我们不应该引入损失或者引入极小的损失。而当 $y_i f(x_i) < 0$ 时, 我们应引入损失, 而且 $y_i f(x_i)$ 的值越小, 引入的损失越大。因此, 这里我们引入指数函数 $\exp(-y_i f(x_i))$ 来处理这两种不同的情况。注意, 当 $y_i f(x_i) \geq 0$ 时, 我们仍然引入了损失, 但是很小。指数函数的最大优点是它是连续可导的, 而且导数非常容易计算, 使得我们在求解相关的优化问题时很容易计算。

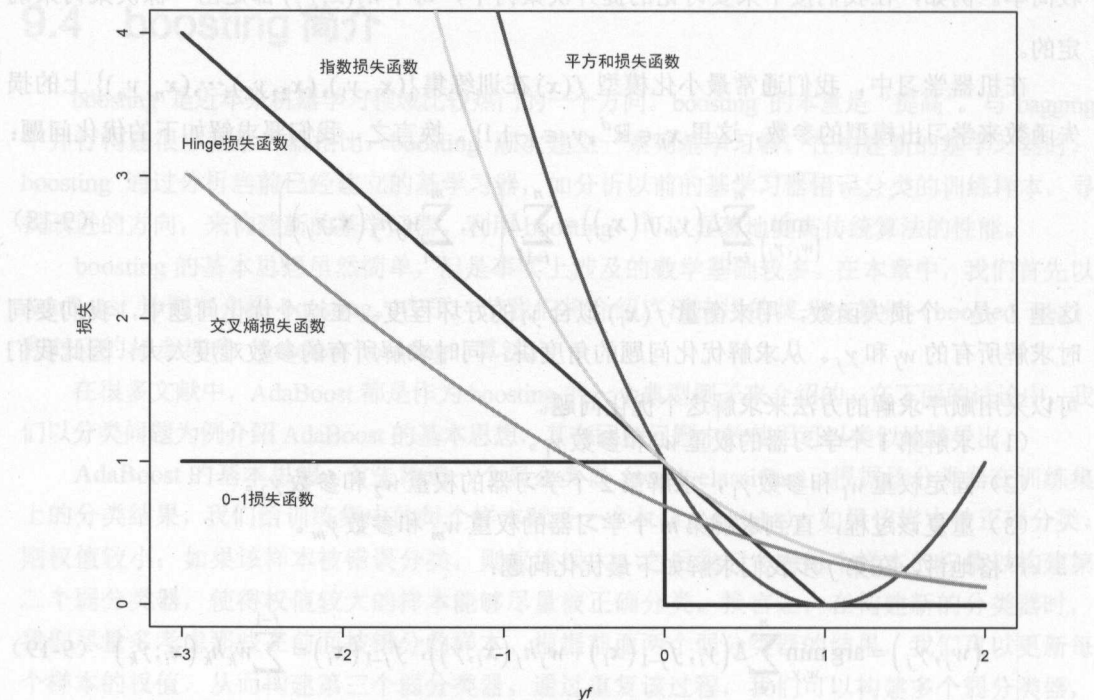


图 9-3 分类算法中的常用损失函数

9.4.2 AdaBoost 算法

事实上, 当使用指数损失函数时, 如果我们在 boosting 中使用顺次求解 (forward stage-wise additive modeling) 的方式来导出权重和分类器, 就得到了著名的 AdaBoost (Adaptive Boosting)

算法。下面我们将进行推导并给出 AdaBoost 算法的具体流程。

假设我们已经推导出前面的 $j-1$ 个学习器的权重和参数，并记前 $j-1$ 个学习器的加权和所得的模型 $f_{j-1}(\mathbf{x})$ 为：

$$f_{j-1}(\mathbf{x}) = \sum_{k=1}^{j-1} w_k h_k(\mathbf{x}) \quad (9-21)$$

在使用指数损失函数的情况下，在第 j 步我们需要求解如下问题：

$$(w_j, h_j) = \arg \min_{w, h} F = \arg \min_{w, h} \sum_{i=1}^n \exp(-y_i (f_{j-1}(\mathbf{x}_i) + wh(\mathbf{x}_i))) \quad (9-22)$$

这里我们将目标值简写为：

$$\begin{aligned} F &= \sum_{i=1}^n \exp(-y_i (f_{j-1}(\mathbf{x}_i) + wh(\mathbf{x}_i))) \\ &= \sum_{i=1}^n \exp(-y_i f_{j-1}(\mathbf{x}_i)) \exp(-y_i wh(\mathbf{x}_i)) \end{aligned} \quad (9-23)$$

注意， $\exp(-y_i f_{j-1}(\mathbf{x}_i))$ 在这里只依赖于 y_i 和 $f_{j-1}(\mathbf{x}_i)$ ，因此在求解第 j 步时为定值，可以将其视为在第 j 步时对于样本 \mathbf{x}_i 的权值。我们将该权值记为：

$$\beta_i^{(j)} = \exp(-y_i f_{j-1}(\mathbf{x}_i)) \quad (9-24)$$

注意，该权值 $\beta_i^{(j)}$ 既与样本 \mathbf{x}_i 的真实类别 y_i 有关，也与已经构建的分类器 f_{j-1} 对该样本的当前分类结果有关。事实上， $\beta_i^{(j)}$ 就是我们使用 $f_{j-1}(\mathbf{x}_i)$ 作为 \mathbf{x}_i 的分类结果所引起的指数损失函数值。因此，目标函数可以简写为：

$$F = \sum_{i=1}^n \exp(-y_i f_{j-1}(\mathbf{x}_i)) \exp(-y_i wh(\mathbf{x}_i)) = \sum_{i=1}^n \beta_i^{(j)} \exp(-y_i wh(\mathbf{x}_i)) \quad (9-25)$$

下面我们讨论在第 j 步如何具体求解权值 w_j 和函数 $h_j(\mathbf{x})$ 。

在分类问题中，我们首先注意到 $\forall j \in [1, m]$ ， $h_j(\mathbf{x}) \in \{-1, 1\}$ ， $y_i \in \{-1, 1\}$ ， $\beta_i^{(j)} > 0$ 。当 $y_i \neq h_j(\mathbf{x}_i)$ 时， $y_i h_j(\mathbf{x}_i) = -1$ ，当 $y_i = h_j(\mathbf{x}_i)$ 时， $y_i h_j(\mathbf{x}_i) = 1$ 。

因此，可以将目标函数表示为：

$$\begin{aligned} F &= \sum_{i=1}^n \beta_i^{(j)} \exp(-y_i wh(\mathbf{x}_i)) \\ &= \sum_{y_i \neq h(\mathbf{x}_i)} \beta_i^{(j)} \exp(w) + \sum_{y_i = h(\mathbf{x}_i)} \beta_i^{(j)} \exp(-w) \end{aligned}$$

$$\begin{aligned}
&= \exp(w) \sum_{y_i \neq h(x_i)} \beta_i^{(j)} + \exp(-w) \sum_{y_i = h(x_i)} \beta_i^{(j)} \quad (9-26) \\
&= \exp(w) \sum_{y_i \neq h(x_i)} \beta_i^{(j)} - \exp(-w) \sum_{y_i \neq h(x_i)} \beta_i^{(j)} + \exp(-w) \sum_{y_i \neq h(x_i)} \beta_i^{(j)} + \exp(-w) \sum_{y_i = h(x_i)} \beta_i^{(j)} \\
&= (\exp(w) - \exp(-w)) \sum_{y_i \neq h(x_i)} \beta_i^{(j)} + \exp(-w) \sum_{i=1}^n \beta_i^{(j)}
\end{aligned}$$

如果假定权重 $w > 0$, 则 $\exp(w) - \exp(-w) > 0, \exp(-w) > 0$ 。注意, 此时 F 中只有前一项中考虑了函数 $h(\mathbf{x})$ 。更进一步, 对于任何一个给定的 $w > 0$, 使得 F 最小的 $h(\mathbf{x})$ 必然使得 $\sum_{y_i \neq h(x_i)} \beta_i^{(j)}$ 最小。换言之, 函数 $h_j(\mathbf{x})$ 可通过求解如下问题得到:

$$h_j(\mathbf{x}) = \arg \min_h \sum_{i=1}^n \beta_i^{(j)} I(y_i \neq h(\mathbf{x}_i)) \quad (9-27)$$

这里函数 $I(x)$ 的定义为:

$$I(x) = \begin{cases} 1, & \text{如果 } x = \text{TRUE} \\ 0, & \text{如果 } x = \text{FALSE} \end{cases}$$

简单地讲, 函数 $h_j(\mathbf{x})$ 就是使得使用权重 $\beta_i^{(j)}$ 之后的错误率最低的分类器。

下面我们推导在给定函数 $h_j(\mathbf{x})$ 的情况下如何得到最优的 w_j 。如果记 $A = \sum_{y_i \neq h_j(x_i)} \beta_i^{(j)}$, $B = \sum_{i=1}^n \beta_i^{(j)}$, 那么可以将目标函数表示为:

$$F = (\exp(w) - \exp(-w))A + \exp(-w)B \quad (9-28)$$

我们计算 F 对 w 的偏导数并设为 0 即可得到最优的 w :

$$\begin{aligned}
\frac{\partial F}{\partial w} &= (\exp(w) + \exp(-w))A - \exp(-w)B = 0 \\
\Rightarrow A \exp(w) &= (B - A) \exp(-w) \\
\Rightarrow \exp(2w) &= \frac{B - A}{A} \\
\Rightarrow w &= \frac{1}{2} \ln \frac{B - A}{A}
\end{aligned} \quad (9-29)$$

如果对于模型 $h_j(\mathbf{x})$, 我们引入加权的错误率 err_j :

$$err_j = \frac{\sum_{y_i \neq h_j(x_i)} \beta_i^{(j)}}{\sum_{i=1}^n \beta_i^{(j)}} = \frac{A}{B} \quad (9-30)$$

则最优的 w_j 可表示为:

$$w_j = \frac{1}{2} \ln \frac{B-A}{A} = \frac{1}{2} \ln \frac{1-err_j}{err_j} \quad (9-31)$$

那么在构建第 j 个分类器后, 前 j 个分类器的和为:

$$f_j(\mathbf{x}) = f_{j-1}(\mathbf{x}) + w_j h_j(\mathbf{x}) \quad (9-32)$$

注意, $\beta_i^{(j)} = \exp(-y_i f_{j-1}(\mathbf{x}_i))$, 那么每个样本对应的权重在构建第 j 个分类器后可以利用如下公式更新:

$$\begin{aligned} \beta_i^{(j+1)} &= \exp(-y_i f_j(\mathbf{x}_i)) \\ &= \exp(-y_i (f_{j-1}(\mathbf{x}_i) + w_j h_j(\mathbf{x}_i))) \\ &= \exp(-y_i f_{j-1}(\mathbf{x}_i)) \exp(-y_i w_j h_j(\mathbf{x}_i)) \\ &= \beta_i^{(j)} \exp(-y_i w_j h_j(\mathbf{x}_i)) \\ &= \beta_i^{(j)} \exp(w_j (2I(y_i \neq h_j(\mathbf{x}_i)) - 1)) \\ &= \beta_i^{(j)} \exp(2w_j I(y_i \neq h_j(\mathbf{x}_i))) \exp(-w_j) \end{aligned} \quad (9-33)$$

在式 (9-33) 中的倒数第二步, 我们利用了性质 $-y_i h_j(\mathbf{x}_i) = 2I(y_i \neq h_j(\mathbf{x}_i)) - 1$ 。验证该式子很简单, 只需要考虑 $y_i h_j(\mathbf{x}_i) = 1$ (即 $y_i = h_j(\mathbf{x}_i)$) 和 $y_i h_j(\mathbf{x}_i) = -1$ (即 $y_i \neq h_j(\mathbf{x}_i)$) 两种不同的情况即可。

这样我们就得到了著名的 AdaBoost 算法。算法 9-3 给出了 AdaBoost 算法的具体步骤。

算法 9-3 AdaBoost 算法

1. 将训练集中每个样本的权值定为 $\beta_i = \frac{1}{n}, i = 1, 2, \dots, n$
2. for $j=1:m$
 - 2.1 构建一个分类器 $h_j(\mathbf{x})$ (考虑训练集中每个样本的权值 β_i)
 - 2.2 计算该分类器加权的错误率

$$err_j = \frac{\sum_{i=1}^n \beta_i I(y_i \neq h_j(\mathbf{x}_i))}{\sum_{i=1}^n \beta_i} \quad (9-34)$$

- 2.3 计算该分类器的权值

$$w_j = \frac{1}{2} \ln \frac{1-err_j}{err_j} \quad (9-35)$$

- 2.4 更新每个样本的权值

$$\beta_i = \beta_i \exp(2w_j I(y_i \neq h_j(\mathbf{x}_i))), i=1, 2, \dots, n \quad (9-36)$$

3. 输出最终的分类器

$$f(\mathbf{x}) = \text{sign} \left(\sum_{j=1}^m w_j h_j(\mathbf{x}) \right) \quad (9-37)$$

在该算法中, 我们顺次构建 m 个分类器, 且每个分类器能够处理样本的权重 (如果分类器不能在训练过程中处理样本的权重, 则我们不能直接应用该分类器)。在第 j 步, 得到该分类器 $h_j(\mathbf{x})$ 后, 我们计算该分类器对应的错误率 err_j 并据此计算 $h_j(\mathbf{x})$ 对应的权重 w_j 。然后根据 $h_j(\mathbf{x})$ 和 w_j 更新每个样本的权重 β_i 。与前面推导的公式相比, 在更新 β_i 时我们省略了 $\exp(-w_j)$, 原因是所有的权重都要乘以 $\exp(-w_j)$, 因此可以直接省略该项。在更新样本的权重 β_i 时, 我们可以看到: 如果 $y_i = h_j(\mathbf{x}_i)$, 则 β_i 不变; 如果 $y_i \neq h_j(\mathbf{x}_i)$, 则 β_i 要乘以权重 $\exp(2w_j)$, 这样在构建下一个分类器时会更加重视这些被错分的样本。因此, 在实际使用 AdaBoost 时, 我们要为每个样本记录它们当前的权重 β_i , 同时也要记录每个分类器对应的权重 w_j 。

9.4.3 AdaBoost 的实际使用

在本节, 我们讨论如何使用 R 中的 `adabag` 包^①来调用 AdaBoost 算法。`adabag` 软件包实现了适用于分类问题的 `boosting` 和 `bagging` 程序。我们提供了 `AdaBoost_classification_example.R` 文件来介绍如何使用 `adabag` 软件包调用 AdaBoost 算法。

在 `adabag` 包中, 需要使用 `boosting` 函数调用 AdaBoost 算法。该函数实现了 AdaBoost 算法, 能够解决两类分类问题和多类分类问题。在使用 `boosting` 函数时, 需要指定数据, 包括具体的输入数据和对应的公式 (指定哪个变量是目标变量, 哪些变量是自变量)。下面我们简要介绍 `boosting` 函数中的几个重要参数。

- 参数 `boos`: 默认值是 `TRUE`, 表示我们将利用当前每个样本的权重使用 `bootstrap` 取样来得到当前的训练集。如果是 `FALSE` 的话, 每个样本以及当前的权重将会被使用 (而不是用 `bootstrap` 取样)。
- 参数 `mfinal`: `boosting` 中弱分类器的总数, 默认值是 100。
- 参数 `coeflearn`: 用来指定在 `boosting` 算法中计算权重 w_j 的方法。可选的值包括 "Breiman" (默认值)、"Freund" 和 "Zhu", 其中 "Breiman" 对应算法 9-3 中 w_j 的更新方法。
- 参数 `control`: 用来控制每个弱分类器的参数。在 `adabag` 包中, 由于我们调用 `rpart` 包训练得到决策树作为弱分类器, 因此使用 `rpart` 包中的 `rpart.control` 函数来具体指定控制参数, 如每棵决策树的深度等。

① <https://cran.r-project.org/web/packages/adabag/index.html>

与前面讨论的标准 AdaBoost 算法相比, 这里的参数 `boos` 指定了在构建每个分类器时利用权值的方法。boosting 函数返回一个 boosting 对象 (也称 boosting, 但是这个是对象, 前一个是函数), 包括所涉及的决策树的数目、每棵决策树的权重, 以及所得的所有决策树的具体参数。我们可以逐一检查返回的决策树。

与随机森林类似, 一棵决策树很容易理解, 但是很多决策树的聚合却不容易理解。在这种情况下, 如果我们能够计算每个变量的重要程度, 将会帮助我们理解所得的模型。利用返回的 boosting 对象和 `errorevol` 函数, 我们可以计算每个变量的重要程度。这里变量的重要程度是利用每棵决策树中每个变量带来的基尼指数的减小, 并综合考虑每棵决策树的权重所得到的。

下面我们通过讲解 `AdaBoost_classification_example.R` 文件中的代码来说明如何使用 `adabag` 包。

首先我们检查 `adabag` 包是否已经安装。如果没有安装, 则首先安装该软件包。在第 1 步, 我们载入 `Sonar` 数据。注意, `mlbench` 包自带该数据集, 而 `adabag` 依赖于 `mlbench` 包。当我们使用 `library(adabag)` 载入 `adabag` 包时, `mlbench` 包也自动载入, 因此, 这里可以使用 `data(Sonar)` 来直接载入该数据集。该数据集的最后一列是类标, 我们将其列名改为 `'classLabel'`。在第 2 步, 我们将数据按照 70% 和 30% 的比例分为训练集和测试集, 并分别保存在数据框 `D_train` 和 `D_test` 中。

```
# Step 0. Check required package is installed or not. If not, install first.
# 1. adabag
adabag.installed <- 'adabag' %in% rownames(installed.packages())
if (adabag.installed) {
  print("the adabag package is already installed, let's load it...")
}else {
  print("let's install the adabag package first...")
  install.packages('adabag', dependencies=T)
}
library('adabag')
library('rpart')

# Step 1. Load the Sonar data in the mlbench library
data(Sonar)
D <- Sonar
colnames(D)[ncol(D)] <- 'classLabel'

# Step 2. Split the data into training and test sets
# Randomly split the whole data set into a training and a test data set
# After splitting, we have the training set: (X_train, y_train)
# and the test data set: (X_test, y_test)
train_ratio <- 0.7
n_total <- nrow(D)
n_train <- round(train_ratio * n_total)
n_test <- n_total - n_train
```



```

set.seed(42)
list_train <- sample(n_total, n_train)
D_train <- D[list_train,]
D_test <- D[-list_train,]
y_train <- D_train$classLabel
y_test <- D_test$classLabel

# Step 3. Benchmark: train a single decision tree using rpart
M_rpart1 <- rpart(classLabel~., data = D_train)
print('show the summary of the trained model')
summary(M_rpart1)

# Compute the performance on the training and test data sets
y_test_pred_rpart1 <- predict(M_rpart1, D_test, type='class')
accuracy_test_rpart1 <- sum(y_test==y_test_pred_rpart1) / n_test
msg <- paste0('accuracy_test_rpart1 = ', accuracy_test_rpart1)
print(msg)

y_train_pred_rpart1 <- predict(M_rpart1, D_train, type='class')
accuracy_train_rpart1 <- sum(y_train==y_train_pred_rpart1) / n_train
msg <- paste0('accuracy_train_rpart1 = ', accuracy_train_rpart1)
print(msg)

# Step 4. Train a simple AdaBoost model
maxdepth <- 4
mfinal <- 60
M_AdaBoost1 <- boosting(classLabel~., data = D_train,
                        boos = FALSE, mfinal = mfinal, coeflearn = 'Breiman',
                        control=rpart.control(maxdepth=maxdepth))

# Check the summary of the trained AdaBoost model
summary(M_AdaBoost1)
# We get all trees
M_AdaBoost1$trees
# print the 1st tree
M_AdaBoost1$trees[[1]]
# We get the weights for all trees
M_AdaBoost1$weights
# We get the variable importance
M_AdaBoost1$importance
# We get the evolution of the error
errorevol(M_AdaBoost1, D_train)
# Check the first tree trained in AdaBoost
t1 <- M_AdaBoost1$trees[[1]]
# Plot the tree
plot(t1, uniform=T, branch=0, margin=0.1, main = 'classification tree')
text(t1, splits=T, all = T, fancy = T)

```

```
# Compute the accuracy of AdaBoost model on the training and test data sets
y_test_pred_AdaBoost1 <- predict(M_AdaBoost1, D_test)
accuracy_test_AdaBoost1 <- sum(y_test==y_test_pred_AdaBoost1$class) / n_test
msg <- paste0('accuracy_test_AdaBoost1 = ', accuracy_test_AdaBoost1)
print(msg)

y_train_pred_AdaBoost1 <- predict(M_AdaBoost1, D_train)
accuracy_train_AdaBoost1 <- sum(y_train==y_train_pred_AdaBoost1$class) / n_train
msg <- paste0('accuracy_train_AdaBoost1 = ', accuracy_train_AdaBoost1)
print(msg)
```

在第 3 步, 我们使用 `rpart` 包构建一个决策树的分类模型以与 AdaBoost 相比较。

在第 4 步, 我们构建了一个 AdaBoost 模型。

```
M_AdaBoost1 <- boosting(classLabel~., data = D_train,
                        boos = FALSE, mfinal = mfinal, coeflearn = 'Breiman',
                        control=rpart.control(maxdepth=maxdepth))
```

在该步骤中, 指定参数 `boos` 为 `FALSE`, 表示我们要使用所有的样本; 将参数 `mfinal` 设为 60, 表示我们将构建 60 棵决策树。此外, 我们还将 `control` 设为 `rpart.control(maxdepth=maxdepth)`, 表示我们使用 `rpart` 包构建决策树时将 `rpart` 函数中的控制参数 `maxdepth` 设为 4。

这样我们就得到了一个称为 `M_AdaBoost1` 的 `boosting` 对象。我们使用 `summary` 函数来输出该 AdaBoost 模型的主要信息。此外, 我们可以使用 `M_AdaBoost1` 中的如下成员来查看该 `boosting` 对象的相关信息。

- `trees`: 得到构建的所有决策树。其中 `trees[[i]]` 表示其中的第 i 棵决策树。
- `weights`: 保存每棵决策树对应的权重。
- `importance`: 保存每个变量的重要程度。

在上面的程序中, 我们使用 `t1` 来保存 `M_AdaBoost1` 中的第一棵决策树, 并使用 `plot` 和 `text` 函数绘出 `t1` 所对应的决策树的图像, 如图 9-4 所示。

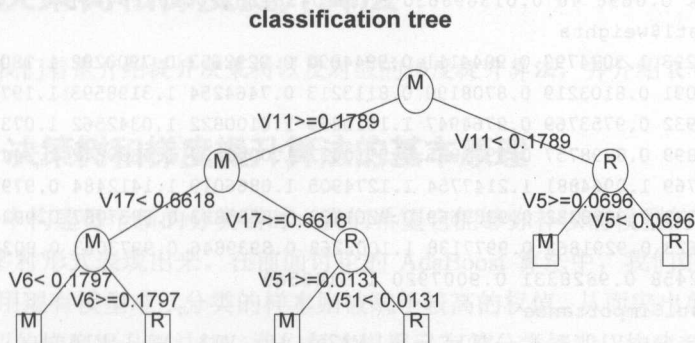


图 9-4 AdaBoost 模型中所得的第一棵树

程序的输出如下:

```

[1] "accuracy_test_rpart1 = 0.693548387096774"
[1] "accuracy_train_rpart1 = 0.876712328767123"
[1] "accuracy_test_AdaBoost1 = 0.854838709677419"
[1] "accuracy_train_AdaBoost1 = 1"

> summary(M_AdaBoost1)
      Length Class Mode
formula      3  formula call
trees        60 -none- list
weights      60 -none- numeric
votes       292 -none- numeric
prob        292 -none- numeric
class       146 -none- character
importance   60 -none- numeric
terms        3  terms  call
call         7  -none- call

> M_AdaBoost1$trees[[1]]
n = 146
node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 146 0.486301400 M (0.52737671 0.47262329)
2) V11>=0.17885 85 0.143835600 M (0.76299475 0.23700525)
4) V17< 0.6618 61 0.047945210 M (0.89069716 0.10930284)
8) V6< 0.17975 54 0.020547950 M (0.94725111 0.05274889) *
9) V6>=0.17975 7 0.020547950 R (0.44204322 0.55795678) *
5) V17>=0.6618 24 0.068493150 R (0.43004587 0.56995413)
10) V51>=0.0131 12 0.020547950 M (0.76013514 0.23986486) *
11) V51< 0.0131 12 0.006849315 R (0.08761682 0.91238318) *
3) V11< 0.17885 61 0.075342470 R (0.18857143 0.81142857)
6) V5>=0.0696 15 0.041095890 M (0.61307902 0.38692098) *
7) V5< 0.0696 46 0.013698630 R (0.04581552 0.95418448) *

> M_AdaBoost1$weights
[1] 0.9808293 1.3074799 0.9044111 0.9944090 0.9292653 0.7900282 1.1807839 1.0509344
[9] 0.9620091 0.8103219 0.8708190 0.8113213 0.7464254 1.3198593 1.1971926 0.9290395
[17] 1.0168932 0.9753769 0.9764947 1.1010859 1.0100822 1.0342562 1.0733267 0.7857949
[25] 1.2606899 0.8798757 0.9931865 0.9910637 1.0070627 1.0809842 1.2443802 1.0938129
[33] 0.8468769 1.0964881 1.2147754 1.1274905 1.0866019 1.1412484 0.9799360 1.0320253
[41] 1.1127173 0.9088232 0.9389669 0.9261423 0.9555823 0.9837087 0.9924242 0.8690865
[49] 0.9102623 0.9291862 0.9977138 1.1003268 0.8939846 0.9973063 0.8033537 1.1148003
[57] 0.9812458 0.9828331 0.9007920 1.3148560

> M_AdaBoost1$importance
      V1      V10      V11      V12      V13      V14      V15      V16
1.4812561 1.5426775 7.8247186 5.3938873 1.5469844 1.1158881 0.4165955 0.8154501
      V17      V18      V19      V2      V20      V21      V22      V23
2.1841351 0.0000000 0.2783136 0.3078752 0.9507150 1.3128143 1.1018555 2.7719996
      V24      V25      V26      V27      V28      V29      V3      V30

```

```

0.1851123 0.0000000 1.0870610 4.5926235 2.0182482 0.0000000 0.2588854 0.0000000
V31      V32      V33      V34      V35      V36      V37      V38
0.8016263 0.4935591 0.5347451 3.7400880 1.3330086 6.4139607 1.1847926 0.2944026
V39      V4      V40      V41      V42      V43      V44      V45
0.4898486 2.7445508 0.0000000 0.4289211 1.9320352 3.1744688 1.3742040 5.4004112
V46      V47      V48      V49      V5      V50      V51      V52
3.5877499 0.3224826 3.2149998 1.7710574 0.9377694 0.2963930 2.1314830 2.6108785
V53      V54      V55      V56      V57      V58      V59      V6
0.7701156 1.0362824 2.5357311 0.6760309 1.2456797 1.7136457 1.6977423 1.6227034
V60      V7      V8      V9
1.5132980 1.1750007 1.3802089 2.2330285

```

在上面的程序中,我们比较了 `rpart` 所构建的决策树模型和 `AdaBoost` 所构建的分类模型在测试集和训练集上的准确率。从上面的输出可以看出,与 `rpart` 所构建的单个决策树相比,`AdaBoost` 所构建的模型在训练集上的准确率从 0.877 提高到了 1.000;同时,在测试集上的准确率从 0.694 提高到了 0.855。这里我们为了显示结果的方便,在使用 `AdaBoost` 时只构建了 60 棵决策树。当我们把决策树的数目提高到 100 时,`AdaBoost` 所得的模型在测试集上的准确率会进一步提高到 0.887,当决策树数目提高到 200 时,测试集上的准确率会提高到 0.903。

9.4.4 讨论

与 `bagging` 相比, `boosting` 更加激进地从训练集中提取信息来训练模型。因此, `boosting` 更容易受到噪声的影响,更容易导致过拟合。举一个简单的例子,如果我们在训练集中将某一样本的类标标错了,那么该样本有可能成为训练集中较难分类的样本,导致我们在使用 `boosting` 的过程中增大了该样本的权重,从而使得训练所得的分类器受到了错误的影响。此外,由于在 `boosting` 中我们要顺次构建模型,在算法的实现中不如容易并行处理的 `bagging` 速度快。

9.5 提升决策树和梯度提升算法

在本节中,我们着重介绍提升决策树以及对应的梯度提升算法,并介绍 R 中相应的 `gbm` 包。

9.5.1 提升决策树和梯度提升算法的基本原理

在 `boosting` 中构建每个新的分类器时,我们希望它能够弥补以前模型的“弱点”。这里的“弱点”可以以多种形式表现出来。在前面讨论的 `AdaBoost` 算法中,我们给每个样本赋予一个权值,而那些用现有模型难以分类的样本则被赋予较高的权值,从而突出前面分类器的“弱点”。在本节介绍的梯度提升算法中,我们每次根据已有的分类情况,构建新的目标值以更好地分类那些“难分类”的样本。具体来说,对于样本 \mathbf{x}_i ,在构建第 j 个学习器时,我们用 t_{ij} 来表示对应的目标值(注意,开始时 $t_{i0} = y_i$)。通过不停地改变 t_{ij} ,使得聚合后的分类器能够更好地分类样本 \mathbf{x}_i 。

在提升决策树中,我们假设每个弱学习器都是一棵决策树。我们知道 AdaBoost 算法对应指数损失函数,而在提升决策树中,我们可以采用不同的损失函数。与机器学习中的很多算法类似,在提升决策树中,给定训练集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, 我们最小化模型 $f(\mathbf{x})$ 对应的损失函数 $L(y, f)$ 来求解最优的模型:

$$\min_f L(y, f) = \sum_{i=1}^n L(y_i, f(\mathbf{x}_i)) \quad (9-38)$$

在提升决策树中,我们假设 $f(\mathbf{x})$ 表示为一系列决策树的和:

$$f(\mathbf{x}) = \sum_{j=1}^m h_j(\mathbf{x}) \quad (9-39)$$

这里我们把第 j 棵决策树对应的函数记为 $h_j(\mathbf{x})$ 。将第 J ($J \leq m$) 步所得的模型 $f_J(\mathbf{x})$ 记为 $f_J(\mathbf{x}) = \sum_{j=1}^J h_j(\mathbf{x})$ 。

在梯度提升算法中,我们顺次建立模型 $h_j(\mathbf{x})$ 。该算法的核心思想是在第 j ($j \leq m$) 步构建模型 $h_j(\mathbf{x})$ 时,采用负梯度 $-\left[\frac{\partial L(y_i, f)}{\partial f}\right]_{f=f_{j-1}}$ 作为 \mathbf{x}_i 的新目标值来训练模型。从数值最优化的角度讲,这种策略近似于数值最优化中的梯度下降算法。下面我们从平方和损失函数对应的残差和数值最优化的角度阐述使用负梯度值 $-\left[\frac{\partial L(y_i, f)}{\partial f}\right]_{f=f_{j-1}}$ 作为新目标值的合理性。

我们首先讨论平方和损失函数对应的残差。假设在第 j 步,我们已得到模型 $f_{j-1}(\mathbf{x})$, 可以计算残差 (residual) $y_i - f_{j-1}(\mathbf{x}_i)$ 。残差表示了真实值和当前的预测值之间的差异。如果能使差异为 0, 则意味着能得到极好的模型。因此,一种朴素的想法是不妨将残差 $y_i - f_{j-1}(\mathbf{x}_i)$ 作为下一个模型中 \mathbf{x}_i 对应的新目标值。

下面我们从数值优化的角度来说明使用残差作为新目标值的合理性。考虑使用平方和损失函数来度量训练集上的损失:

$$L(y, f) = \sum_{i=1}^n L(y_i, f) = \frac{1}{2} \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2 \quad (9-40)$$

我们要优化 $f(\mathbf{x})$ 使得损失函数 $L(f)$ 最小。注意,这里要优化的对象 $f(\mathbf{x})$ 是一个函数。换言之,我们需要在函数空间中求解该优化问题。如果只考虑训练集,为了简化讨论,可以将 $f(\mathbf{x})$ 考虑为一个 n 维向量 $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n)]^T = [f^{(1)}, f^{(2)}, \dots, f^{(n)}]^T$ 。这样的话,我们需要在 n 维向量空间中寻找使得损失函数 $L(y, \mathbf{f})$ 最小的向量 \mathbf{f} 。

这样我们就可以直接使用数值最优化中的结果。在数值优化中,假设要优化的参数是 \mathbf{f} (这里 $\mathbf{f} \in \mathbb{R}^n$, 为 n 维向量), 要最小化的目标函数是 $L(y, \mathbf{f})$ 。在梯度下降算法的每一步,假

设 f 的当前估计为 \hat{f} ，我们计算其对应的梯度 $\nabla f = \frac{\partial L}{\partial f}$ ，然后使用如下公式来更新 f ：

$$\hat{f} \leftarrow \hat{f} - \nabla f \quad (9-41)$$

当使用平方和损失函数 $L(y, f)$ 时，可以计算 $L(y, f)$ 对于 f 中每个分量 $f^{(i)}$ 的偏导数：

$$\nabla f^{(i)} = \frac{\partial L(y, f)}{\partial f^{(i)}} = f(x_i) - y_i \quad (9-42)$$

写成向量的形式，负梯度可以表示为：

$$-\nabla f = \begin{bmatrix} -\nabla f^{(1)} \\ \vdots \\ -\nabla f^{(n)} \end{bmatrix} = \begin{bmatrix} y_1 - f(x_1) \\ \vdots \\ y_n - f(x_n) \end{bmatrix} \in \mathbb{R}^n \quad (9-43)$$

在第 j 步更新模型 $f(x)$ 时， $f(x_i)$ 的当前值是 $f_{j-1}(x_i)$ 。此时的负梯度 $-\nabla f$ 值为：

$$-\left[\frac{\partial L(y_i, f)}{\partial f} \right]_{f=f_{j-1}} = \begin{bmatrix} y_1 - f_{j-1}(x_1) \\ \vdots \\ y_n - f_{j-1}(x_n) \end{bmatrix} \in \mathbb{R}^n \quad (9-44)$$

这里我们把第 j 步的负梯度记为 $-\left[\frac{\partial L(y_i, f)}{\partial f} \right]_{f=f_{j-1}}$ 。可以看出，在使用平方和损失函数的情况下，负梯度等于我们前面讨论的第 j 步的残差，也是在第 j 步训练新模型 $h_j(x)$ 时的目标值。

我们可以将以上的讨论推广到一般情况。对于不同的问题，我们需要选择不同的损失函数（而不局限于平方和损失函数）。对于不同的损失函数，可以计算相应的负梯度，并使用负梯度作为新的目标值来训练新的学习器。在介绍 R 中的 `gbm` 包时，会涉及不同的损失函数。

注意，在梯度提升算法中，采用负导数 $-\left[\frac{\partial L(y_i, f)}{\partial f} \right]_{f=f_{j-1}}$ 作为新的目标来构建模型，我

们只用到了训练集中的数据来计算导数，并最小化训练集上的损失函数。但是在机器学习中，我们希望得到的模型不但要在训练集上表现优秀，更重要的是能够很好地处理训练集之外的数据（即我们希望所得模型的泛化能力要好）。因此，在实际中，我们要尽量避免在训练集上的过拟合现象。通常我们采用一类函数（在提升决策树中是回归决策树）来拟合负导数，并不要求完美拟合。

算法 9-4 梯度提升算法

1. 得到初始函数 $f_0(x) = c$ ，这里 $c = \arg \min_c \sum_{i=1}^n L(y_i, c)$
2. for $j=1:m$
 - 2.1 计算负梯度：

$$r_{ij} = - \left[\frac{\partial L(y, f)}{\partial f^{(i)}} \right]_{f=f_{j-1}} \quad (9-45)$$

2.2 以 $\mathbf{r} = [r_{1j}, r_{2j}, \dots, r_{nj}]^T$ 作为新的目标, 在训练集上构建一个新模型 $h_j(\mathbf{x})$

2.3 求解步长 s_j :

$$s_j = \arg \min_{s>0} \sum_{i=1}^n L(y_i, f_{j-1}(\mathbf{x}_i) + sh_j(\mathbf{x}_i)) \quad (9-46)$$

2.4 将函数 f 更新为:

$$f_j(\mathbf{x}) = f_{j-1}(\mathbf{x}) + s_j h_j(\mathbf{x}) \quad (9-47)$$

3. 输出最终的模型:

$$f(\mathbf{x}) = f_m(\mathbf{x}) \quad (9-48)$$

算法 9-4 给出了梯度提升算法的具体步骤。在第一步中, 我们求解一个恒量 c 来最小化损失函数 $\sum_{i=1}^n L(y_i, c)$ 作为初始模型 $f_0(\mathbf{x})$ 。

在算法 9-4 的 2.3 步, 我们还进一步求解最优的步长 s_j 以更快地最小化损失函数。我们可以把 $\sum_{i=1}^n L(y_i, f_{j-1}(\mathbf{x}_i) + sh_j(\mathbf{x}_i))$ 看成一个关于 s 的函数 $F(s)$, 找出使得 $F(s)$ 最小的 s 。简单的实现包括找出使得导数 $F'(s) = 0$ 的 s 。在第 8 章介绍 LambdaMART 算法时, 我们使用牛顿近似 (即二阶泰勒展式) 来逼近 $F(s) \approx F_0 + F'(s)s + \frac{1}{2}F''(s)s^2$, 这里 $F'(s)$ 和 $F''(s)$ 是函数 $F(s)$ 关于 s 的一阶导数和二阶导数。注意, $F_0 + F'(s)s + \frac{1}{2}F''(s)s^2$ 是关于 s 的二次函数, 使得其最小的 s 为 $s = -\frac{F'(s)}{F''(s)}$ 。

注意, 如果我们选择的每个模型 $h_j(\mathbf{x})$ 是决策树的话, 则在 2.3 步求解步长时, 可以进一步对新决策树 $h_j(\mathbf{x})$ 的每个叶结点求解最优的步长以进一步最小化损失函数。我们知道, 当使用训练集来训练得到一棵决策树时, 每个叶结点都对应了一组训练样本。假设我们的决策树 $h_j(\mathbf{x})$ 有 n_j 个叶结点, 将第 l 个叶结点所对应的训练样本的集合记为 R_{jl} , 则可以求出其对应的权重 s_{jl} :

$$s_{jl} = \arg \min_s \sum_{\mathbf{x}_i \in R_{jl}} L(y_i, f_{j-1}(\mathbf{x}_i) + sh_j(\mathbf{x}_i)) \quad (9-49)$$

与之相对应的是, 在 2.4 步更新模型时, 我们要为不同的样本赋予不同的权重。具体来说, 更新公式如下:

$$f_j(\mathbf{x}) = f_{j-1}(\mathbf{x}) + \sum_{l=1}^{n_j} s_{jl} I(\mathbf{x} \in R_{jl}) \quad (9-50)$$

这里函数 $I(\mathbf{x} \in R_{jl})$ 的定义为:

$$I(\mathbf{x} \in R_{jl}) = \begin{cases} 1, & \text{如果 } \mathbf{x} \in R_{jl} \\ 0, & \text{否则} \end{cases} \quad (9-51)$$

9.5.2 如何避免过拟合

下面我们讨论实际使用梯度提升算法时的一些技巧。利用这些技巧,可以显著地提高算法的性能,同时能有效地避免过拟合。

从理论上讲,在梯度提升算法中,我们可以无限制地构建新的决策树以优化分类模型的性能。与机器学习中的很多算法类似,如果对模型的复杂度不加以控制的话,则很容易导致过拟合。在梯度提升算法中,主要通过如下途径避免过拟合。

- (1) 控制学习率;
- (2) 控制决策树的总数;
- (3) 控制每棵决策树的大小;
- (4) 子取样。

下面逐一讨论这些防止过拟合的措施。

1. 学习率和决策树的数目

在提升决策树中,我们构建了多棵决策树。从直观上讲,最开始构造的决策树更多地描述了最后所得模型的主要框架;而后面的决策树更多地是考虑那些难分类的样本。因此,我们可以认为应该给前面的决策树更大的权重。而随着更多的决策树的建立,我们应该逐渐降低其权重。具体来说,在建立第 j 个模型时,我们使用如下的公式来更新模型:

$$f_j(\mathbf{x}) = f_{j-1}(\mathbf{x}) + \nu h_j(\mathbf{x}) \quad (9-52)$$

这里参数 ν 满足 $0 < \nu < 1$, 是新的决策树 $h_j(\mathbf{x})$ 的权重。如果我们把这个公式和数值最优化中的更新公式相比较,可以把 ν 看成是沿着负梯度方向移动的步长,因此 ν 也可以认为是控制了 boosting 算法的学习率。另外,我们可以将学习率与我们前面反复讨论的模型的正则化相联系。事实上,梯度提升算法的发明者 Jerome Friedman 在这方面就有相关的讨论,这里我们就不深入讨论了。

在实际使用梯度提升算法时,决策树数目和学习率是相互依赖的。一般而言,较小的 ν 意味着我们要以较慢的速度去拟合目标函数,需要构建较多的决策树;较大的 ν 则意味着可以构建较少的决策树。在实际中,较小的学习率一般能够得到更好的模型。举一个简单的例子,将 ν 设置为 0.005,在 OOB 样本中的表现一般来说要显著强于将 ν 设置为 0.05 时的情况。但是为了得到较好的性能,较小的 ν 值就会要求较多的决策树,这就要求更多的存储空间和计算时间。如果将 ν 从 0.05 降到 0.005,基本上需要多近 10 倍的计算时间。

在实际使用 R 中的 gbm 包时,两个最重要的参数是决策树的数目和学习率,它们在 gbm 包的 gbm 函数中分别对应于参数 n.trees 和 shrinkage。图 9-5 显示了一个在回归问题中,当使

用不同的 shrinkage 值时, 模型在 OOB 样本上的 RMSE 随着 n.trees 的增长而发生的变化。从图 9-5 中可以看出, 在保证 n.trees 足够大的情况下, 降低 shrinkage 的值能够提高模型的性能。当 shrinkage 降低到一定程度之后, 模型性能的提升就不明显了。从图 9-5 中还可以看出, 当我们将 shrinkage 从 0.01 降到 0.005 时, 最优的 n.trees 的值也差不多增加了一倍, 但是最优的 RMSE 只是稍稍减少。因此, 一般的经验: 在计算时间允许的条件下, 设置尽量小的 shrinkage 值, 再设置合适的 n.trees 的值。我们一般倾向于较小的 ν 值 ($\nu < 0.1$)。

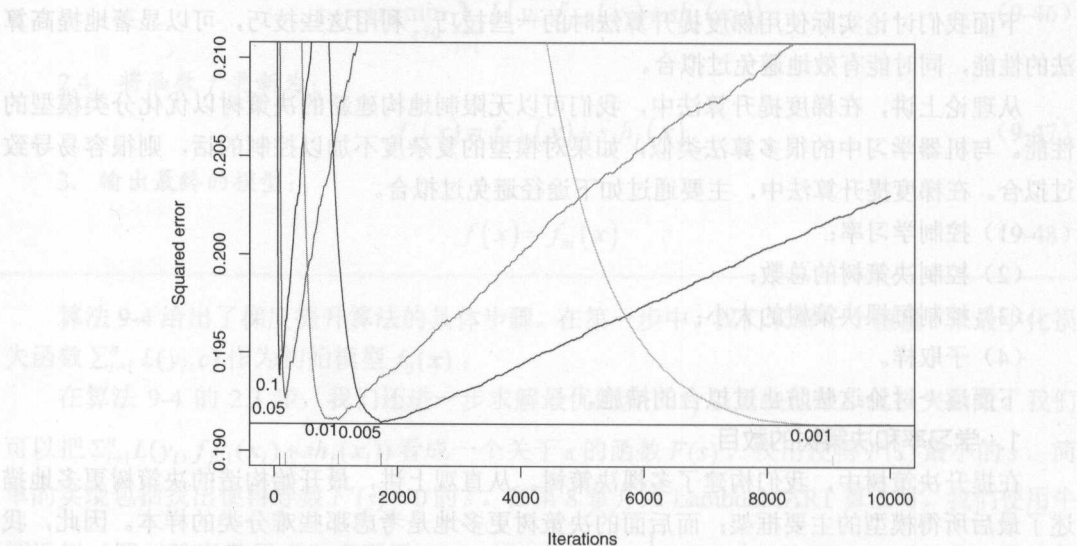


图 9-5 gbm 中学习率 (shrinkage) 和决策树数目 (n.trees) 的关系^① (每条曲线对应于一个学习率的值, 横坐标是决策树的数目, 纵坐标是 OOB 样本上的 RMSE 值)

2. 决策树的大小

控制决策树的大小可以通过控制决策树的深度或者叶结点的数目来实现。由于在梯度提升算法中我们构建了多棵决策树, 因此我们一般倾向于每棵决策树的复杂度不宜太高。例如, 我们可以限制每棵决策树的叶结点的数为一恒定值, 或者每棵决策树的深度为一恒定值。而这些都可以作为梯度提升算法的参数, 可根据实际的数据予以调节。

3. 子取样

子取样 (subsampling) 的思想与 bootstrap 取样类似。在随机森林中, 通过 bootstrap 取样显著降低了生成的决策树之间的相关性。在梯度提升算法中, 也可以采用类似的方法。

具体来说, 在使用子取样时, 我们从所有的样本中随机取出一部分 (但不是重复取样) 的样本。这样, 每次构建新的决策树时, 只通过子取样得到一部分训练样本来构建新的模型。通过使用子取样, 能够在构建决策树的时候引入一些随机性, 从而能够增强各个模型的多样性, 进而提高聚合之后的模型的性能。在 R 的 gbm 包中, 子取样是通过参数 bag.fraction

^① 本图来源于: Greg Ridgeway, Generalized Boosted Models: A guide to the gbm package.

来控制的。在实践中，每次子取样时我们使用一半左右的样本，这在很多时候都被证明是一个行之有效的推荐值。当样本量大时，我们还可以进一步降低每次取样时所得样本的数目。

4. 更加完备和实用的梯度提升算法

综合考虑多种防止过拟合的措施，我们将得到更加完备和实用的梯度提升算法。算法 9-5 列出了在 R 中常用的 gbm 包中实现的梯度提升算法。事实上，在实际使用 gbm 包时，还有更多的输入参数，在算法 9-5 中我们仅列出了最主要的控制参数及具体步骤。

算法 9-5 gbm 包中实现的梯度提升算法

输入：

- 损失函数的形式（参数 distribution）
- 决策树的数目 m （参数 n.trees）
- 决策树的叶结点数 K （由参数 interaction.depth 决定）
- 学习率 v （参数 shrinkage）
- 抽样比例 p （参数 bag.fraction）

算法的具体步骤如下。

1. 得到初始函数 $f_0(\mathbf{x}) = c$ ，这里 $c = \arg \min_c \sum_{i=1}^n L(y_i, c)$

2. for $j=1:m$

2.1 计算负梯度：

$$r_{ij} = - \left[\frac{\partial L(y, f)}{\partial f^{(i)}} \right]_{f=f_{j-1}} \quad (9-53)$$

2.2 从原始数据集中随机选出 pn 个样本（总样本数为 n ）

2.3 以 $\mathbf{r} = [r_{1j}, r_{2j}, \dots, r_{nj}]^T$ 作为新的目标，使用 2.2 步中选出的训练集，训练得到一个决策树 T_j ，且该决策树的叶结点数是 K

2.4 使用当前选出的训练集，为 T_j 的每个叶结点求出对应的权重 s_{jl} 。将第 l 个叶结点所对应的训练样本的集合记为 R_{jl} ，则 s_{jl} 为：

$$s_{jl} = \arg \min_s \sum_{\mathbf{x}_i \in R_{jl}} L(y_i, f_{j-1}(\mathbf{x}_i) + s h_j(\mathbf{x}_i)) \quad (9-54)$$

2.5 将函数 f 更新为：

$$f_j(\mathbf{x}) = f_{j-1}(\mathbf{x}) + v \sum_{l=1}^K s_{jl} I(\mathbf{x} \in R_{jl}) \quad (9-55)$$

3. 输出最终的模型：

$$f(\mathbf{x}) = f_m(\mathbf{x}) \quad (9-56)$$

9.5.3 gbm 包的的实际使用

这里我们介绍 R 中最流行的实现了梯度提升算法的 gbm 包^①。在 gbm 包中,采用的是决策树作为基本的弱学习器。利用 gbm 包中的 gbm 函数,可以使用梯度提升算法来构建分类、回归和排序模型。gbm 函数的主要参数设置如下:

- 损失函数的形式 (distribution);
- 决策树的数目 (n.trees);
- 决策树的内结点数目 (interaction.depth);
- 学习率 (shrinkage);
- 子取样比例 (bag.fraction)。

下面我们具体介绍这些参数并给出一些参数设置的建议,并使用实际数据来说明 gbm 函数的用法。

首先我们要根据问题的类型来决定损失函数的形式。损失函数是我们首先需要确定的参数。在上面的讨论中,我们使用了平方和损失函数来进行推导。事实上,很多其他的损失函数都可以在梯度提升算法中使用。gbm 包中提供的适用于分类问题的常用损失函数有:

- 'bernoulli'
- 'adaboost'
- 'huberized'
- 'multinomial'

其中 'bernoulli' 使用交叉熵损失函数, 'adaboost' 是前面讨论过的指数损失函数, 'huberized' 是 huberized Hinge 损失函数 (Hinge 损失函数的一种变体)。前面 3 个都适用于两类分类问题。而 'multinomial' 对应于多类分类问题的损失函数。一般而言,对于两类分类问题,我们使用 'bernoulli' 和 'adaboost' 就足够了。

对于回归问题,可以将损失函数设为:

- 'gaussian'
- 'laplace'
- 'quantile'

其中, 'gaussian' 表示损失函数是平方和损失函数, 'laplace' 表示损失函数是预测值与真实值的差的绝对值之和:

$$L(y, f) = \sum_{i=1}^n |y_i - f(\mathbf{x}_i)| \quad (9-57)$$

这里假设我们的预测值是 $f(\mathbf{x}_i)$, 而真实值是 y_i 。当选用 'quantile' 时,我们使用分位数回归 (quantile regression) 所对应的损失函数。一般而言,在回归问题中,我们采用 'gaussian'

① <https://cran.r-project.org/web/packages/gbm/index.html>

可适用于大部分问题。

gbm 包还能处理生存分析 (survival analysis)。在这种情况下, 需要将 `distribution` 设为 `'coxph'`。此外, 我们也可以使用 gbm 包来解决排序问题。在排序相关章节我们已经作了详细介绍, 这里就不再讨论了。

在默认情况下, 如果不指定 `distribution` 参数的值, gbm 会根据数据的特征来确定 `distribution` 的值。如果输入数据的目标变量只有两个不同的值, 则认为问题是分类问题, 并将 `distribution` 的值设为 `'bernoulli'`。如果输入数据的目标变量有多于两个不同的值, gbm 再检查输入数据的目标变量的类型: 如果是因子类型, 就将 `distribution` 的值设为 `'multinomial'`; 如果不是因子类型, gbm 自动检查问题是否是生存分析问题; 如果都不是的话, 将默认问题为回归问题并将 `distribution` 的值设为 `'gaussian'`。

在决定损失函数后, 接下来最重要的参数是学习率 (`shrinkage`) 和决策树的数目 (`n.trees`)。一般的经验为: 在计算时间允许的条件下, 设置尽量小的 `shrinkage` 值, 再设置合适的 `n.trees` 的值。一般来讲, 可以将 `shrinkage` 的参数设置在 0.01~0.001。对应地, 可以将 `n.trees` 设置在 3000 至 10000 之间。

在设定了 `shrinkage` 的值后, 如何设置决策树的数目? gbm 包给出了 3 种估计最优的决策树的数目的方法, 分别为:

- 利用训练集;
- 利用 OOB 中的样本;
- 利用交叉检验。

核心思想都是利用某一个数据集来估计决策树的数目。具体来说, 在 gbm 包中, 我们可以使用 `gbm.perf` 函数来计算估计决策树的数目。

对于每棵决策树的大小, gbm 中使用参数 `interaction.depth` 来控制。该参数指定了每棵决策树中内结点的数目。根据内结点的数目, 可以相应地控制叶结点的数目。此外, 利用 gbm 包中的函数 `pretty.gbm.tree`, 可以提取 gbm 所构建的决策树中的某棵决策树的对应信息。

我们提供了文件 `gbm_classification_example.R`, 用它介绍如何使用梯度提升算法来解决分类问题。具体的代码如下。

首先我们检查 gbm 包有没有安装, 如果没有安装, 那么首先安装它。这里我们使用 `mlbench` 包中的 `PimaIndiansDiabetes` 数据集, 所以也要安装 `mlbench` 包。

然后我们导入数据, 并检查每列的类型。之后我们将数据按照 70% 和 30% 的比例分为训练集和测试集, 并分别保存在数据框 `D_train` 和 `D_test` 中。

```
# Step 0. Check required package is installed or not. If not, install first.
# 1. gbm
gbm.installed <- 'gbm' %in% rownames(installed.packages())
if (gbm.installed) {
  print("the gbm package is already installed, let's load it...")
}else {
  print("let's install the gbm package first...")
}
```



```

install.packages('gbm', dependencies=T)
}
library(gbm)
# 2. mlbench
mlbench.installed <- 'mlbench' %in% rownames(installed.packages())
if (mlbench.installed) {
  print("the mlbench package is already installed, let's load it...")
}else {
  print("let's install the mlbench package first...")
  install.packages('mlbench', dependencies=T)
}
library(mlbench)

# Step 1. Load the data
data(PimaIndiansDiabetes2, package='mlbench')
D <- PimaIndiansDiabetes2
y <- D[[ncol(D)]]
y <- as.integer(y) - 1
D[[ncol(D)]] <- NULL
D$classLabel <- y

# Show the type for each col
for(i in 1:ncol(D)) {
  msg <- paste('col ', i, ' and its type is ', class(D[,i]))
  print(msg)
}

# Step 2. Split the data into training and test sets
# Randomly split the whole data set into a training and a test data set
# After splitting, we have the training set: (X_train, y_train)
# and the test data set: (X_test, y_test)
train_ratio <- 0.7
n_total <- nrow(D)
n_train <- round(train_ratio * n_total)
n_test <- n_total - n_train
set.seed(42)
list_train <- sample(n_total, n_train)
D_train <- D[list_train,]
D_test <- D[-list_train,]

y_train <- D_train$classLabel
y_test <- D_test$classLabel

# Step 3. Train several gbm models
# Train a simple gbm model
M_gbm1 <- gbm(classLabel~.,
               data = D_train,
               distribution = 'bernoulli',

```

```

        shrinkage = 0.01,
        interaction.depth = 1,
        n.trees = 300,
        verbose = T)
print('the summary of M_gbm1 is')
print(M_gbm1)

# Expand the gbm model by adding more trees
M_gbm1.1 <- gbm.more(M_gbm1, n.new.trees = 100)
print('the summary of M_gbm1.1 is')
print(M_gbm1.1)

# Train a gbm model using cross-validation
set.seed(1)
M_gbm2 <- gbm(classLabel~.,
               data = D_train,
               distribution='bernoulli',
               shrinkage = 0.01,
               n.trees=3000,
               cv.folds = 5,
               verbose=F)
print('the summary of M_gbm2 is')
print(M_gbm2)

# We use gbm.perf to get the estimate of the optimal number of trees using
# cross-validation
best.iter2 <- gbm.perf(M_gbm2,method = 'cv')
msg <- paste0('the best n.tree is ', best.iter2)
print(msg)
# Show the variable importance using the first best.iter2 trees
varImp2 <- summary(M_gbm2, best.iter2, main = 'variable importance of M_gbm2')
# Show the marginal effect of the selected variables by "integrating" out the
# other variables
# Here we select the 3rd variable.
plot.gbm(M_gbm2, 3, best.iter2)

# compactly print the first and last trees for curiosity
print('the structure of the 1st tree')
print(pretty.gbm.tree(M_gbm2,1))
print('the structure of the last tree')
print(pretty.gbm.tree(M_gbm2, M_gbm2$n.trees))

# predict the new data using the "best" number of trees
y_test_pred_gbm2 <- predict(M_gbm2, D_test, best.iter2, type = 'response')
print('we are going to print the first few predictions of y_test_pred_gbm2')
print(head(y_test_pred_gbm2))
y_test_pred_gbm2_raw <- predict(M_gbm2, D_test, best.iter2, type = 'link')
print('we are going to print the first few predictions of y_test_pred_gbm2_raw')

```

```

print(head(y_test_pred_gbm2_raw))

# Train a more complicated model
M_gbm3 <- gbm(classLabel~.,
              data = D_train,
              distribution='bernoulli',
              shrinkage = 0.005,
              bag.fraction = 0.4,
              cv.folds = 5,
              interaction.depth = 2,
              n.cores = 4,
              n.trees=3000,
              verbose=F)

print('the summary of M_gbm3 is')
print(M_gbm3)

# Step 4. Use gbm to do multi-class classification
data(iris)
M_iris <- gbm(Species ~ .,
              distribution="multinomial",
              data=iris,
              n.trees=2000,
              shrinkage=0.01,
              cv.folds=5,
              verbose=F,
              n.cores=1)

print('the summary of M_iris is')
print(M_iris)
print('the variable importance of M_iris')
summary(M_iris, main = 'variable importance of M_iris')

```

在第3步，我们构建了多个 gbm 模型。由于这里使用的数据对应两类分类问题，因此将 distribution 设为 'bernoulli'。注意，当 distribution 为 'bernoulli' 时，我们必须保证对应于类标的列的值必须是 0 或者 1。这也是我们在第一步对 y 进行预处理的原因。使用如下代码我们构建了第一个 gbm 模型 M_gbm1：

```

M_gbm1 <- gbm(classLabel~., data = D_train, distribution = 'bernoulli',
              shrinkage = 0.01, interaction.depth = 1, n.trees = 300, verbose = T)

```

上述参数的含义我们在前面已经解释过了。我们在这里将参数 verbose 设为真，表示 R 将会打印模型构建中的一些信息。然后，我们使用 print(M_gbm1) 打印出该模型的主要信息，则对应的输出为：

```

Iter  TrainDeviance  ValidDeviance  StepSize  Improve
1      1.2772         nan         0.0100    0.0025
2      1.2722         nan         0.0100    0.0024
3      1.2675         nan         0.0100    0.0022

```

4	1.2624	nan	0.0100	0.0022
5	1.2580	nan	0.0100	0.0021
6	1.2534	nan	0.0100	0.0020
7	1.2480	nan	0.0100	0.0020
8	1.2439	nan	0.0100	0.0021
9	1.2401	nan	0.0100	0.0020
10	1.2361	nan	0.0100	0.0020
20	1.1983	nan	0.0100	0.0015
40	1.1375	nan	0.0100	0.0012
60	1.0962	nan	0.0100	0.0008
80	1.0626	nan	0.0100	0.0006
100	1.0338	nan	0.0100	0.0005
120	1.0106	nan	0.0100	0.0005
140	0.9901	nan	0.0100	0.0003
160	0.9720	nan	0.0100	0.0004
180	0.9552	nan	0.0100	0.0002
200	0.9410	nan	0.0100	0.0002
220	0.9279	nan	0.0100	0.0000
240	0.9169	nan	0.0100	0.0002
260	0.9069	nan	0.0100	0.0002
280	0.8969	nan	0.0100	-0.0000
300	0.8877	nan	0.0100	0.0001

```
[1] "the summary of M_gbm1 is"
```

```
gbm(formula = classLabel ~ ., distribution = "bernoulli", data = D_train,
     n.trees = 300, interaction.depth = 1, shrinkage = 0.01, verbose = T)
```

```
A gradient boosted model with bernoulli loss function.
```

```
300 iterations were performed.
```

```
There were 8 predictors of which 8 had non-zero influence.
```

Iter	TrainDeviance	ValidDeviance	StepSize	Improve
301	0.8874	nan	0.0100	-0.0001
302	0.8871	nan	0.0100	-0.0001
303	0.8868	nan	0.0100	-0.0001
304	0.8863	nan	0.0100	0.0000
305	0.8859	nan	0.0100	0.0001
306	0.8855	nan	0.0100	-0.0001
307	0.8853	nan	0.0100	-0.0001
308	0.8848	nan	0.0100	0.0001
309	0.8843	nan	0.0100	0.0000
310	0.8838	nan	0.0100	0.0000
320	0.8799	nan	0.0100	0.0001
340	0.8722	nan	0.0100	-0.0000
360	0.8658	nan	0.0100	-0.0000
380	0.8588	nan	0.0100	0.0001
400	0.8528	nan	0.0100	0.0000

在实际中,如果当前的 gbm 模型不理想,那么可能要进一步增加决策树的数目。在 gbm 模型中,我们不必从头开始重新构建 gbm 模型。我们可以使用 gbm.more 函数增加当前 gbm 模型中决策树的数目。

```
M_gbm1.1 <- gbm.more(M_gbm1, n.new.trees = 100)
```

这里我们又增加了 100 棵决策树,而且将新的决策树模型记为 M_gbm1.1。同样地,我们也可以使用 print 函数打印新 M_gbm1.1 的主要信息。其输出如下:

```
[1] "the summary of M_gbm1.1 is"
gbm.more(object = M_gbm1, n.new.trees = 100)
A gradient boosted model with bernoulli loss function.
400 iterations were performed.
There were 8 predictors of which 8 had non-zero influence.
```

接下来我们介绍如何使用 gbm.perf 函数来估计最优的决策树的数目。首先我们使用交叉检验得到模型 M_gbm2:

```
M_gbm2 <- gbm(classLabel~., data = D_train, distribution='bernoulli', shrinkage
= 0.01, n.trees=3000, cv.folds = 5, verbose=F)
```

与前面的不同是,我们将 cv.folds 设为 5,表示在训练中采用 5 重交叉检验。接下来我们调用 gbm.perf 就可以得到利用交叉检验得到的最优决策树的数目。

```
best.iter2 <- gbm.perf(M_gbm2,method = 'cv')
```

这段代码的输出为:

```
[1] "the summary of M_gbm2 is"
gbm(formula = classLabel ~ ., distribution = "bernoulli", data = D_train,
     n.trees = 3000, shrinkage = 0.01, cv.folds = 5, verbose = F)
A gradient boosted model with bernoulli loss function.
3000 iterations were performed.
The best cross-validation iteration was 774.
There were 8 predictors of which 8 had non-zero influence.
[1] "the best n.tree is 774"
```

可以看出,最优的决策树的数目是 774。注意,gbm.perf 函数同时也绘出了损失函数在训练集和检验集上随着决策树增加而引起的变化,如图 9-6 所示。从图 9-6 中可以看出,当增加决策树的数目时,训练集对应的损失函数一直在单调降低,而检验集上对应的损失函数先降低再上升。

接下来我们使用 summary 函数得到 gbm 模型对应的参数的重要程度。

```
varImp2 <- summary(M_gbm2, best.iter2, main = 'variable importance of M_gbm2')
```

该函数除了返回一个数据框外,还绘图显示了各个变量的重要程度,如图 9-7 所示。

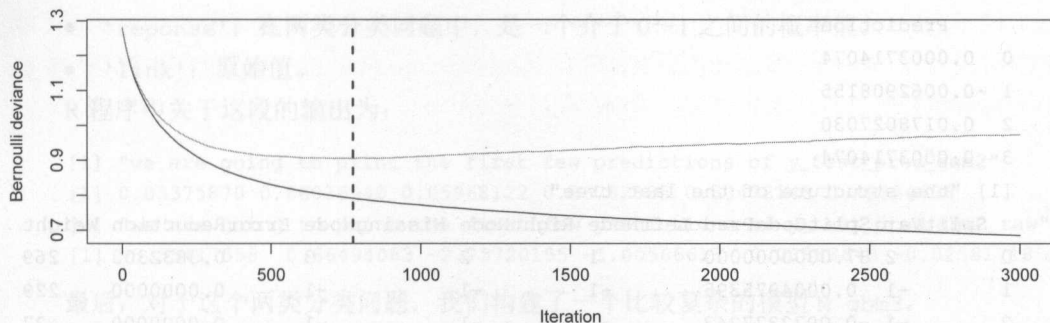


图 9-6 gbm 中所使用的损失函数随着决策树数目的增加而引起的变化（下面的曲线对应于训练集，上面的曲线对应于检验集，竖直的虚线代表我们选取的最优的决策树的数目的位置）

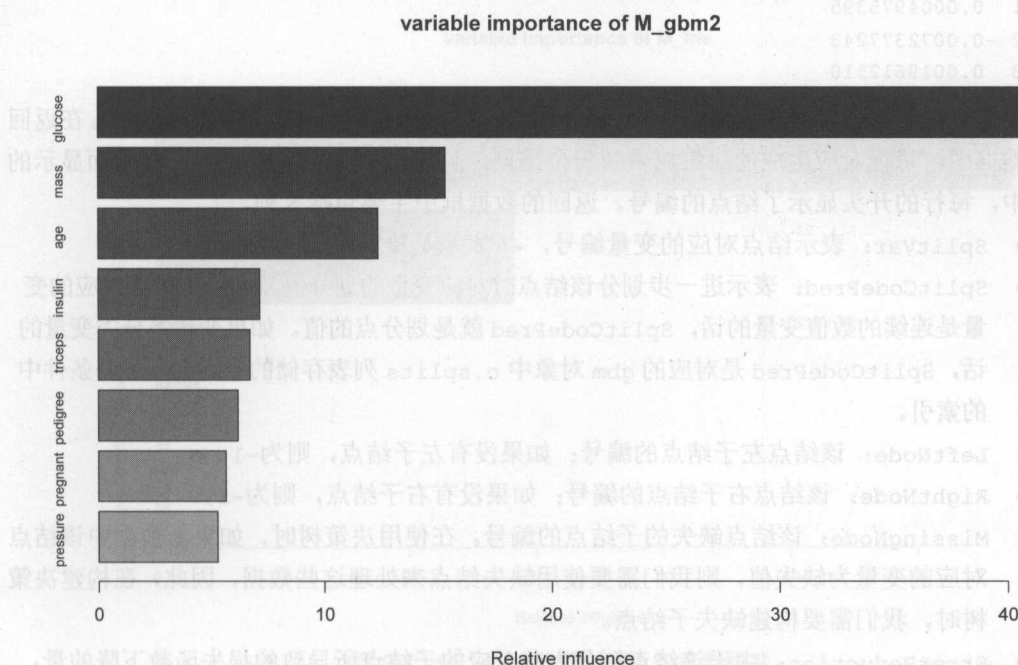


图 9-7 M_gbm2 模型中变量的重要程度

接下来我们使用 `pretty.gbm.tree(M_gbm2, i)` 来取得 gbm 模型 M_gbm2 中第 i 棵树中变量重要性的信息。下面是第一棵和最后一棵树对应的变量重要信息：

```
[1] "the structure of the 1st tree"
  SplitVar SplitCodePred LeftNode RightNode MissingNode ErrorReduction Weight
0         1  1.390000e+02         1         2           3       15.45783      269
1        -1 -6.290815e-03        -1        -1        -1         0.00000      191
2        -1  1.780270e-02        -1        -1        -1         0.00000       73
3        -1  3.714074e-04        -1        -1        -1         0.00000        5
```

```

      Prediction
0  0.0003714074
1 -0.0062908155
2  0.0178027030
3  0.0003714074

[1] "the structure of the last tree"

  SplitVar SplitCodePred LeftNode RightNode MissingNode ErrorReduction Weight
0       2 87.00000000000      1         2           3      0.3832303      269
1      -1  0.0004975395      -1        -1          -1      0.0000000      229
2      -1 -0.0072377243      -1        -1          -1      0.0000000      27
3      -1  0.0019612510      -1        -1          -1      0.0000000      13

      Prediction
0 -0.0002081254
1  0.0004975395
2 -0.0072377243
3  0.0019612510

```

函数 `pretty.gbm.tree` 返回一个数据框以显示 `gbm` 生成的某一棵树的具体结构。在返回的数据框中，每行对应生成的决策树中的一个结点，且结点的编号从 0 开始。在上面显示的结果中，每行的开头显示了结点的编号。返回的数据框中主要包括 8 列。

- **SplitVar**: 表示结点对应的变量编号，-1 表示是决策树的叶结点。
- **SplitCodePred**: 表示进一步划分该结点时对应变量的划分值。如果该结点对应的变量是连续的数值变量的话，`SplitCodePred` 就是划分点的值。如果变量是分类变量的话，`SplitCodePred` 是对应的 `gbm` 对象中 `c.splits` 列表存储的分类变量划分条件中的索引。
- **LeftNode**: 该结点左子结点的编号；如果没有左子结点，则为 -1。
- **RightNode**: 该结点右子结点的编号；如果没有右子结点，则为 -1。
- **MissingNode**: 该结点缺失的子结点的编号。在使用决策树时，如果新数据中该结点对应的变量为缺失值，则需要使用缺失结点来处理这些数据，因此，在构建决策树时，我们需要构建缺失子结点。
- **ErrorReduction**: 由于该结点划分为其对应的子结点所导致的损失函数下降的量；如果该结点是叶结点，则 `ErrorReduction` 为 0。
- **Weight**: 该结点对应的所有样本点的总权重。如果每个样本的权重都是 1，则 `Weight` 为该结点对应的样本的数目。
- **Prediction**: 表示在该结点没有被进一步划分之前，我们应该赋给该结点所有样本点的最优目标值。

在上面第一个棵树的输出中，我们可以看出有 4 个结点，标号为 0~3，其中第一个结点是根结点（其左子结点为结点 1，右子结点为结点 2），后面 3 个结点都是叶结点（`SplitVar` 为 -1）。

这里我们要特别说明，在使用 `predict` 函数（实质上我们调用的是 `predict.gbm` 函数）处理 `gbm` 对象时，需要指定以下参数 `type` 的值。

- 'reponse': 在两类分类问题中, 是一个介于 0~1 之间的概率值。
- 'link': 原始值。

R 程序中关于这段的输出为:

```
[1] "we are going to print the first few predictions of y_test_pred_gbm2"
[1] 0.03375870 0.66036940 0.05968122 0.26782885 0.50472148 0.49354606
[1] "we are going to print the first few predictions of y_test_pred_gbm2_raw"
[1] -3.35417558 0.66494083 -2.75720155 -1.00566613 0.01888648 -0.02581718
```

最后, 对于这个两类分类问题, 我们构建了一个比较复杂的模型 M_{gbm3} 。

在第 4 步中, 我们使用 `gbm` 在 `iris` 数据集上为多类分类问题构建了一个模型, 主要区别在于将参数 `distribution` 设为了 "multinomial"。我们也使用 `summary` 函数打印变量的重要程度信息, 结果如图 9-8 所示。

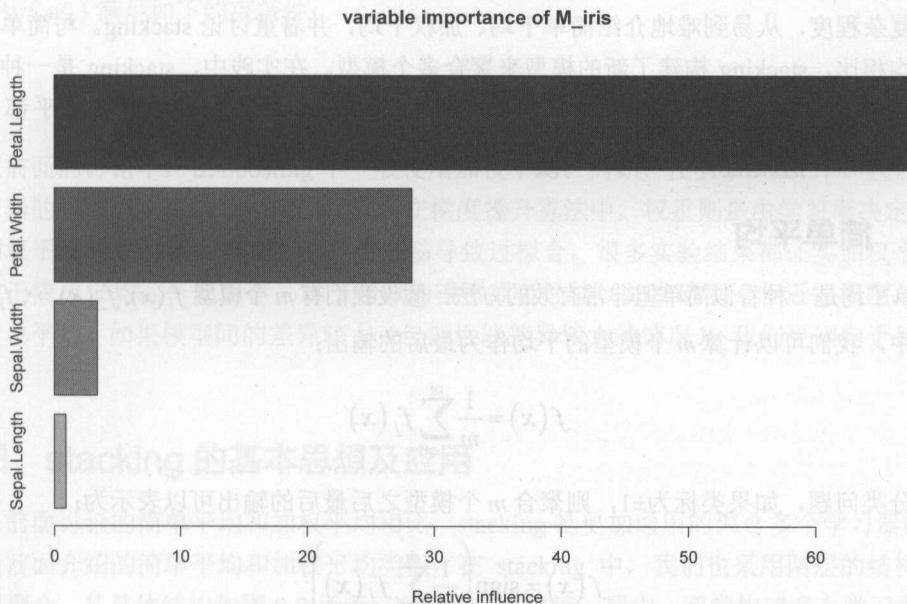


图 9-8 M_{iris} 模型中变量的重要程度

9.5.4 讨论

与 AdaBoost 相比, 在梯度提升算法中, 我们通过使用负梯度为新的目标值来构建后继的模型从而处理那些难以分类的样本; 而在 AdaBoost 中, 我们直接给每个样本赋予了一个权值, 而权值的大小直接反映了下面构建的模型对每个点的重视程度。一般而言, 对于一般的数据集, 在随机森林中, 我们都要构建数百棵决策树。而对于 boosting, 要取得同样的性能, 所需的决策树的数目更多。但是如果仔细选择参数, 一般而言, 当达到 boosting 所需的决策树数目后, boosting 的性能能够优于随机森林。我们的经验为: 对于随机森林的使用, 简单的参数

设置就能得到较好的性能；而对于 boosting，对于很多数据集，我们都需要仔细调节参数才能得到较好的性能。我们推荐读者使用 R 中的 randomForest 包和 gbm 包来学习如何使用随机森林和梯度提升算法，从而获得第一手的使用经验。

9.6 学习器的聚合及 stacking

在本节中，我们讨论如何将多个学习器的结果聚合以得到更好的结果。在前面的讨论中，我们假设每个基学习器为弱学习器，如在随机森林和提升决策树中的基学习器都是决策树。在本节的讨论中，则不限于弱学习器。特别是在一些注重算法性能の場合，通常会构建多个不同类型的学习器。例如，对于一个给定的分类问题，可以构建多个模型，如随机森林、SVM 和逻辑回归；然后再将这 3 个模型的结果聚合以得到最终的模型。在本节中，我们根据方法的复杂程度，从易到难地介绍简单平均、加权平均，并着重讨论 stacking。与简单平均和加权平均相比，stacking 构建了新的模型来聚合多个模型。在实践中，stacking 是一种行之有效的聚合多个模型的方法，如在很多数据挖掘的竞赛中，最终获胜的方案几乎都是使用 stacking 聚合多个模型得到的。

9.6.1 简单平均

简单平均是一种看似简单但非常有效的方法。假设我们有 m 个模型 $f_1(x), f_2(x), \dots, f_m(x)$ 。在回归中，我们可以计算 m 个模型的平均作为最后的输出：

$$f(x) = \frac{1}{m} \sum_{j=1}^m f_j(x) \quad (9-58)$$

在分类问题，如果类标为 ± 1 ，则聚合 m 个模型之后最后的输出可以表示为：

$$f(x) = \text{sign} \left(\frac{1}{m} \sum_{j=1}^m f_j(x) \right) \quad (9-59)$$

这里 sign 是符号函数。在前面的讨论中，随机森林就是使用简单平均来聚合多个弱学习器的。

该方法简单易行。在实际中，很多场合通过使用简单平均就能够显著提高性能。因此，简单平均是最常用的综合多种算法的方法。在很多实际应用中，简单平均是聚合多个模型的第一选择，也是其他复杂聚合方法的第一比较对象。

当然，也存在一些极端情况使得简单平均不能取得较好的性能。例如，如果各个模型 $f_j(x)$ 相互关联且关联度很高，则简单平均不易取得较好的性能。一个简单的极端例子：

$$f_1(x) = f_2(x) = \dots = f_m(x) \quad (9-60)$$

在这种情况下，使用简单平均后所得的模型与原来的 m 个模型相比，性能没有任何提高。

9.6.2 加权平均

加权平均是简单平均的进一步扩展。在加权平均中，我们使用如下公式来聚合多个分类器：

$$f(\mathbf{x}) = \sum_{j=1}^m w_j f_j(\mathbf{x}) \quad (9-61)$$

这里 w_j 是第 j 个模型的权重。事实上，简单平均是加权平均的特例，只需要将权重设为：

$$w_1 = w_2 = \dots = w_m = \frac{1}{m} \quad (9-62)$$

在一些应用中，我们还可以对 $\{w_1, w_2, \dots, w_m\}$ 添加新的约束条件。例如，可以要求 $\{w_1, w_2, \dots, w_m\}$ 满足如下约束条件：

$$w_j \geq 0, \sum_{j=1}^m w_j = 1 \quad (9-63)$$

在前面的讨论中，在 boosting 中一般使用加权平均。例如，在 AdaBoost 中，我们根据每个学习器的错误率来决定其相应的权重。在梯度提升算法中，权重则是由学习率决定的。

加权平均的表达能力更强，但是更容易导致过拟合。很多实验结果都证实加权平均并不一定优于简单平均。一般而言，对于相似的模型或者性能（如准确率）相近的模型，我们倾向于简单平均；如果模型间的差异较大（包括性能差异较大的情况），我们更倾向于使用加权平均。

9.6.3 stacking 的基本思想及应用

与前面讨论的简单平均和加权平均相比，stacking 是更加通用的聚合多个学习器的方法。

与前面介绍的简单平均和加权平均类似，在 stacking 中，我们也采用两层结构来将多个模型聚合。其具体结构如图 9-9 所示。在 stacking 的第一层中，通常构建多个学习器，并将这一层的学习器称为第一层学习器（first-level learner）。在得到第一层学习器后，我们将这一层的输出作为新的“特征”，重新训练一个学习器，称为第二层学习器（second-level learner）。简而言之，stacking 就是将已有的学习器的输出作为新的特征，再训练一个学习器以进一步优化性能。在 stacking 的第二层中，我们可以选择多种不同的算法，如一些非线性算法（如人工神经网络等）。但在 stacking 的实践中，在第二层采用复杂的模型很容易引起过拟合。因此，很多时候，在第二层中我们一般倾向于比较简单的模型，如线性回归。

在这里，我们要着重指出，在 stacking 中第一层的学习器并不局限于弱学习器。事实上，在采用 stacking 时，第一层学习器一般都是性能较好的模型，如随机森林等。为了提高第一层学习器的性能，我们要求第一层学习器之间存在多样性。

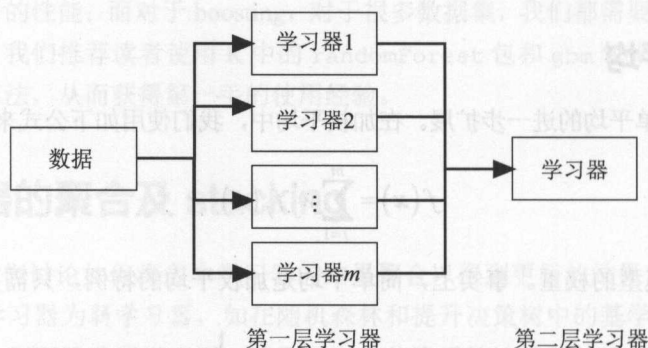


图 9-9 stacking 中的第一层学习器和第二层学习器

在使用 stacking 时，特别需要注意的是，在训练第二层学习器的时候不要使用第一层学习器已经使用过的训练数据。如图 9-10 所示，假设我们使用训练集 S_1 来训练第一层学习器。在得到第一层的模型后，我们计算这些模型在新的训练集 S_2 上的输出，并将这些输出作为新的特征。利用新的训练集 S_2 ，我们训练第二层学习器。得到所有的第一层学习器和第二层学习器后，我们可以得到最后的模型在测试集 T 上的输出，并作为最终输出。注意，如果 S_1 和 S_2 是同一数据集的话，那么在实际中很难避免过拟合。

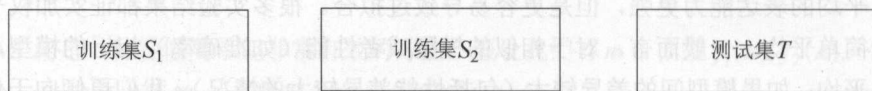


图 9-10 训练数据和测试数据的划分

为了充分利用训练集中的信息，我们还可以在 stacking 中使用交叉检验。具体来说，我们可将整个训练集 S 等分成 k 个子集： S_1, S_2, \dots, S_k ，并记 $S_{-p} = S \setminus S_p$ 为 S_p 的补，如图 9-11 所示。

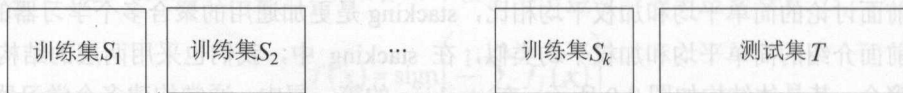


图 9-11 使用交叉检验来应用 stacking 时训练数据和测试数据的划分

这里我们考虑建立 m 个不同的模型 $f_1(x), f_2(x), \dots, f_m(x)$ 。对于每个模型 $f_j(x)$ ，我们使用如下流程来得到 $f_j(x)$ 在整个训练集上的输出。对于 $p \in \{1, 2, \dots, k\}$ ，我们使用 S_{-p} 作为训练集，使用 S_p 作为测试集。我们将这样训练得到的模型记为 $f_j^{(-p)}$ 。这样，对于测试集 S_p 中的每个样本 x_i ，我们可以计算 $f_j^{(-p)}$ 对应的输出。具体来说，对于 $x_i \in S_p$ ，我们记 $z_{ij} = f_j^{(-p)}(x_i)$ 。这样，当 p 值从 1 增长到 k 、 j 从 1 增长到 m 后，交叉检验结束，我们将得到 m 个模型 $f_1(x), f_2(x), \dots, f_m(x)$ 在整个训练集上的输出，记为 S_{cv} ：

$$S_{cv} = \left\{ (z_{11}, z_{12}, \dots, z_{1m}, y_1), (z_{21}, z_{22}, \dots, z_{2m}, y_2), \dots, (z_{n1}, z_{n2}, \dots, z_{nm}, y_n) \right\} \quad (9-64)$$

这里 y_i 是样本 \mathbf{x}_i 对应的类标或者目标值。

利用这个新的训练集，我们可以构建第二层学习器 h 。当得到第二层学习器后，我们重新回到第一层学习器：利用整个训练集 S 重新训练所有的第一层学习器 $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})$ ，然后利用刚才得到的第二层学习器 h 得到最终的结果。

在回归问题中，我们可以直接将第一层模型 $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})$ 的输出作为特征。在分类问题中，我们既可以将第一层模型的输出作为特征，也可以将分类对应的概率作为特征（如果第一层模型可以输出概率的话，如逻辑回归）。

为何 stacking 能够提高第一层学习器的性能？与前面的 bagging 类似，stacking 也利用不同学习器之间存在的多样性。这样第一层学习器之间就能够取长补短，有效提高学习器的综合性能。因此，我们一般在第一层构建不同类型的模型。或者，在构建每个第一层学习器时，我们每次都是用不同的数据集，如使用不同样本的子集，或者不同特征子集。

但是与 bagging 和 boosting 相比，在实际使用中 stacking 更容易导致过拟合。当 stacking 使用不当时，其性能甚至低于简单平均。在大多数情况下，在 stacking 中我们一般不推荐使用特别复杂的第二层模型。事实上，线性回归在很多时候就能够有效提高第一层学习器的性能。

9.7 小结

在实际使用集成学习时，bagging 方法最终得到的模型几乎总是比单个模型效果要好。一般来讲，boosting 比 bagging 对噪声更敏感，更易于过拟合。因此，在实际使用 boosting 时，要仔细调节参数以避免过拟合。通过选择合适的参数，boosting 的效果可以明显好于 bagging 和单个模型。另外，bagging 和 boosting 主要是聚合弱学习器，而 stacking 主要是聚合一些不那么“弱”的学习器。一般来讲，stacking 适合在建模的最后阶段聚合已有的多个模型。

在集成学习中，一个很重要的概念是随机性。在随机森林中，我们通过 bootstrap 取样和在特征空间随机选择特征子集以引入随机性。在梯度上升算法中，我们也通过子取样来引入随机性。事实上，在 stacking 中，也可以采用类似的方法来引入随机性。为什么随机性重要呢？原因是在很多机器学习算法中，我们都是通过贪心学习（greedy learning）的方法从数据中学习模型的参数。贪心学习的好处是可以很快地收敛到局部最优点，坏处就是容易导致过拟合。因此，在集成学习中，通过引入随机性，我们希望能够消灭或者降低贪心学习带来的过拟合风险。

参考文献

- [1] MITCHELL T M. Machine learning [M]. New York: McGraw-Hill, 1997.
- [2] HASTIE T, TIBSHIRANI R, FRIEDMAN J. The elements of statistical learning: data mining, inference, and prediction [M]. 2nd ed. New York: Springer, 2009.
- [3] JAMES G, WITTEN D, HASTIE T, TIBSHIRANI R. An introduction to statistical learning: with applications in R [M]. New York: Springer, 2013.
- [4] BISHOP C M. Pattern recognition and machine learning [M]. New York: Springer, 2006.
- [5] MURPH K P. Machine learning: a probabilistic perspective [M]. Cambridge: MIT Press, 2012.
- [6] KUHN M, JOHNSON K. Applied predictive modeling [M]. New York: Springer, 2013.
- [7] LANTZ B. Machine learning with R [M]. 2nd ed. Birmingham: Packt Publishing, 2015.
- [8] DUDA R O, HART P E, STORK D G. Pattern classification [M]. New York: John Wiley & Sons, 2000.
- [9] TAN P, STEINBACH M, KUMAR V. Introduction to data mining [M]. New York: Pearson Education, 2006.
- [10] HAN J, Pei J, KAMBER M. Data mining: concepts and techniques [M]. 3rd ed. Burlington: Morgan Kaufmann, 2012.
- [11] WITTEN I, FRANK E, HALL M. Data mining: practical machine learning tools and techniques [M]. 3rd ed. Burlington: Morgan Kaufmann, 2011.
- [12] LESKOVEC J, RAJARAMAN A, ULLMAN J. Mining of massive datasets [M]. Cambridge: Cambridge University Press, 2014.
- [13] BECKER R A, CHAMBERS J M. Design of the S system for data analysis [J]. AT&T Technical Journal, 1985, 64(9): 2131-2151.
- [14] BECKER R A, CHAMBERS J M, WILKS A R. The new S language: a programming environment for data analysis and graphics [J]. Economic Journal, 1988, -1(401): 85-89.
- [15] Data Analysis Products Division. S-plus programmer's guide (version 4.5) [M]. MathSoft Inc, 1998.
- [16] SUSSMAN G J, STEELE G L. An interpreter for extended lambda calculus [M]. Cambridge: MIT Press, 1975.
- [17] IHAKA R, GENTLEMAN R. R: a language for data analysis and graphics [J]. Journal of Computational & Graphical Statistics, 1996, 5(3): 299-314.

- [18] VENABLES W N, SMITH D M, R Core Team. An introduction to R [G/OL]. [2016-10-31]. <https://cran.rstudio.com/doc/manuals/r-release/R-intro.html>.
- [19] R Core Team. R language definition [G/OL]. [2016-10-31]. <https://cran.rstudio.com/doc/manuals/r-release/R-lang.html>.
- [20] R Core Team. Writing R extensions [G/OL]. [2016-10-31]. <https://cran.rstudio.com/doc/manuals/r-release/R-lang.html>.
- [21] GOLUB G H, VAN LOAN C F. Matrix computations [M]. 4th ed. Baltimore: Johns Hopkins University Press, 2013.
- [22] PETERSEN K B, PEDERSEN M S. The matrix cookbook [M]. Denmark: Technical University of Denmark, 2012.
- [23] WICKHAM H. ggplot2: Elegant graphics for data analysis [M]. 2nd ed. New York: Springer, 2016.
- [24] GOLUB G H, VAN LOAN C F. Matrix computations [M]. 3rd ed. Baltimore: Johns Hopkins Press, 1996.
- [25] TIBSHIRANI R. Regression shrinkage and selection via the Lasso [J]. Journal of the Royal Statistical Society, Series B, 1996, 58(1): 267-288.
- [26] VAPNIK V. The nature of statistical learning theory [M]. New York: Springer, 2000.
- [27] BOYD S, VANDENBERGHE L. Convex optimization [M]. New York: Cambridge University Press, 2004.
- [28] LESKOVEC J, RAJARAMAN A, ULLMAN J. Mining of massive datasets [M]. Cambridge: Cambridge University Press, 2011.
- [29] SHASHUA A, LEVIN, A. Ranking with large margin principles: two approaches [C]. //Advances in Neural Information Processing Systems. 2002: 937-944.
- [30] BURGESS C, SHAKED T, RENSHAW E, et al. Learning to rank using gradient descent [C]. //International Conference on Machine Learning. ACM, 2005: 89-96.
- [31] BURGESS C, RAGNO R, LE Q V. Learning to rank with non-smooth cost functions [C]. //Advances in Neural Information Processing Systems, 2006: 193-200.
- [32] WU Q, BURGESS C, SVORE K, GAO J. Adapting boosting for information retrieval measures [J]. Information Retrieval Journal, 2010, 13(3): 254-270.
- [33] TSOCHANTARIDIS I, HOFMANN T, JOACHIMS T, ALTUN Y. Large margin methods for structured and interdependent output variables [J]. Journal of Machine Learning Research, 2005: 6(Sep): 1453-1484.

索引

- “间隔和”策略, 263
- 0-1 分布, 42
- AdaBoost 算法, 305
- AFM 模型, 216, 217
- apply 函数族, 34
- ASVD 模型, 216, 218
- bagging, 285, 331
- boosting, 300
- bootstrap 取样, 286
- c()函数, 18
- cat()函数, 18
- cbind(A,B,...), 26
- colMeans(A), 26
- colSums(A), 26
- crossprod(A, B), 26
- csv 文件, 29, 76
- DCG, 257, 260
- det(A), 26
- diag(A), 26, 68
- diag(a,k), 68
- diag(k), 68
- diag(v), 68
- dim(A), 26
- eigen(A), 26, 69
- Elastic Net, 114
- F1 度量, 195
- getRatingMatrix(), 240
- getRatings(), 240
- ggplot2, 89
- glmnet 包, 122
- glmnet()函数, 123
- head(A,k), 26
- help()函数, 17
- Heritage Health Prize 竞赛, 3
- Hinge 损失函数, 166
- item, 205
- Jaccard 相似度, 227
- k 近邻, 128
- LambdaRank 算法, 271
- Lasso, 110
- length(x), 22
- ls()函数, 17
- MAP, 257, 258
- max(x), 22
- mean(x), 22
- median(x), 22
- min(x), 22
- Mini-batch 算法, 270
- ncol(A), 26
- NDCG, 257, 261
- Netflix Prize, 250
- nratings(), 240
- nrow(A), 26
- paste()函数, 17
- paste0()函数, 17
- Pearson 相关系数, 227, 228
- plot()函数, 18
- predict()函数, 123
- prod(x), 22
- range(x), 22
- rbind(A,B,...), 26
- rev(x), 22
- rm()/remove()函数, 17
- rowMeans(A), 26
- rowSums(A), 26
- R 软件包, 36
- Sigmoid 函数, 148
- sort(x), 22
- Spearman 秩相关系数, 227
- stacking, 328, 329
- sum(x), 22
- t(A), 26
- tail(A,k), 26
- tcrossprod(A, B), 26
- unique(x), 22
- which(x>a), 22
- which.max(x), 22
- which.min(x), 22

- Z 分值标准化, 83
- 奥卡姆剃刀, 138
- 半监督型学习, 2
- 半正定矩阵, 55
- 包, 36, 180
- 贝叶斯公式, 41
- 变量, 74
- 标量, 19
- 标量型, 206
- 标准包, 36
- 标准差, 47
- 标准化, 110, 223
- 表达式语句, 14, 15
- 并行方法, 284
- 伯努利分布, 42
- 不对称因子模型, 217
- 不平衡分类, 9, 201
- 布尔型, 206
- 残差, 312
- 残差平方和, 115
- 测试集, 3, 128
- 查询级别的标准化, 266
- 长条图, 92
- 长尾分布, 207
- 超级赋值, 19
- 词频-逆文档频率, 210
- 大数据, 1
- 代价敏感学习, 201, 203
- 单位矩阵, 55
- 等距分组, 88
- 等频分组, 88
- 低偏差, 288
- 第二层学习器, 329
- 第二类错误, 194
- 第一层学习器, 329
- 第一类错误, 194
- 定间隔策略, 263
- 对称矩阵, 54
- 对角矩阵, 54
- 对偶问题, 168
- 对象, 15, 74, 127
- 多类分类, 7, 127
- 多项式核, 172
- 多样性, 284
- 二分变量, 75
- 二维均匀分布, 45
- 二项分布, 42, 47
- 翻转的 AFM 模型, 216, 217, 218
- 翻转的 ASVD 模型, 216, 220
- 泛型, 36
- 方差, 47, 224, 287
- 非监督型学习, 2
- 分布函数, 43
- 分层抽样, 190
- 分类, 6, 104, 127, 262
- 分类变量, 23, 74, 86
- 分类泛化, 163
- 分类间隔, 162
- 分类问题, 306
- 分量, 29
- 分位数, 50
- 分位数回归, 318
- 分组, 88
- 复数型, 20
- 复杂度参数, 109
- 高方差, 288
- 高斯核, 172
- 高斯-马尔可夫定理, 107
- 割平面算法, 281
- 格搜索, 174
- 公式对象, 34
- 贡献包, 36
- 贡献率, 67
- 关联规则, 2, 8
- 广义线性模型, 148
- 国际数据挖掘大会, 5
- 过滤, 225
- 过拟合, 117, 284
- 函数调用语句, 14, 16
- 行列式, 55
- 行秩, 55
- 核, 161, 164
- 核方法, 161
- 核函数, 171
- 核技巧, 172
- 回归, 6, 104, 262
- 回归平方和, 115
- 混淆矩阵, 193
- 机器学习, 1
- 基本函数, 17
- 基本类型, 19
- 基尼指数, 132, 133

- 基于记忆的推荐算法, 222
- 基于矩阵分解的推荐算法, 207
- 基于邻域的推荐算法, 207, 222
- 基于内容的推荐算法, 206, 210
- 基于平均值的标准化, 223
- 基于启发式的推荐算法, 222
- 基于商品的邻域推荐算法, 225
- 基于协同过滤的推荐算法, 206
- 基于用户的邻域推荐算法, 223
- 基准算法, 211
- 极差, 50
- 极大似然估计, 53, 149
- 集成学习, 8, 10
- 几何分布, 42, 47
- 监督型学习, 2, 254
- 剪枝, 138
- 检验集, 180, 216
- 交叉检验, 9, 124, 180
- 交叉熵损失函数, 150, 267, 268, 265
- 交替算法, 214
- 接收者操作特征曲线, 196
- 解释变量, 3
- 经验损失, 254
- 茎叶图, 95
- 精确率, 195, 209
- 矩阵, 19
- 矩阵的定义, 24
- 矩阵的运算, 25
- 聚类分析, 3, 8
- 决策树, 130
- 决策树算法, 132
- 绝对平均偏差, 51, 84
- 均方差, 114
- 均方根误差, 114, 209
- 均匀分布, 42, 47
- 可决系数, 114
- 可逆矩阵, 55
- 控制语句, 14, 16
- 拉格朗日乘子, 168
- 类标, 127, 262
- 累积贡献率, 67
- 冷启动, 211
- 离差平方和, 115
- 离群数据, 88
- 连接函数, 148
- 两类分类, 7, 104, 127
- 列表, 19, 29
- 列秩, 55
- 灵敏度, 194
- 岭回归, 108
- 逻辑回归, 148
- 逻辑型, 20
- 名称变量, 75
- 模式, 3
- 模式识别, 1
- 模型, 3, 115, 287
- 模型复杂度, 9
- 模型选择, 180
- 目标变量, 3
- 牛顿法, 151
- 排序学习, 253
- 偏差, 287
- 偏差-方差权衡, 9, 115, 117, 287
- 偏度, 47, 51
- 平方和损失函数, 105
- 平均反击中率, 209
- 平均精度, 258
- 平均绝对误差, 209
- 朴素贝叶斯分类器, 8
- 奇异值分解, 61, 212
- 启发式方法, 85
- 前 k 个文档的精度, 258
- 欠拟合, 117
- 强化学习, 2
- 取样, 96
- 全概率公式, 41
- 缺失值, 76
- 人工智能, 1
- 软件包的安装, 37
- 软件包的使用, 38
- 弱分类器, 300
- 弱学习器, 284, 328
- 散点图, 100
- 熵, 132
- 上取样, 202
- 深度学习, 7

- 生存分析, 319
- 事后剪枝, 138
- 事前剪枝, 138
- 数据点, 3, 74
- 数据矩阵, 75
- 数据科学, 13
- 数据可视化, 89
- 数据框, 19, 26
- 数据框的操作, 27
- 数据框的定义, 26
- 数据探索, 8, 75
- 数据挖掘, 1, 2
- 数据预处理, 8, 82
- 数据质量, 76
- 数学期望, 46
- 数值变量, 74
- 数值型, 20, 206
- 数值优化, 1
- 数组, 19
- 水平, 23
- 顺次求解, 302
- 顺序变量, 75
- 顺序方法, 284
- 四分位极差, 51
- 似然函数, 53
- 随机变量, 39
- 随机猜测, 284
- 随机森林, 285, 289
- 随机试验, 39
- 随机梯度下降算法, 10, 211, 212, 267, 214
- 随机向量, 43
- 损失函数, 9, 105, 127, 175
- 贪心学习, 331
- 特异度, 194
- 特征, 74
- 特征工程, 8
- 特征向量, 57
- 特征值, 57
- 特征值分解法, 65
- 梯度提升算法, 273, 300, 311, 312
- 梯度下降算法, 274, 312
- 提升决策树, 274, 300, 311
- 条状图, 92
- 统计学, 1
- 投票, 286
- 凸替代, 178
- 凸优化, 214
- 推荐系统, 205
- 维数灾难, 170
- 文档检索, 253
- 文档频率, 255
- 稀疏解, 111
- 下标系统, 33
- 下取样, 202
- 显著性, 230
- 显著性权值, 230
- 线性核, 173
- 线性回归, 105
- 线性模型, 105
- 相关系数, 100
- 相似度, 227
- 箱线图, 96
- 向量, 19, 74
- 向量的范数, 56
- 向量的运算, 22
- 协同过滤, 205
- 信息检索, 253
- 信息增益, 134, 135, 290
- 信用分数, 4
- 修正的余弦相似度, 229
- 学习率, 215, 315
- 学习器, 3
- 训练集, 3, 128, 180
- 哑变量, 87, 120
- 样本, 3, 39, 74
- 样本空间, 39
- 一元值型, 206
- 因变量, 3, 104
- 因子, 19, 23, 120
- 引入时间信息的矩阵分解模型, 216
- 隐式反馈, 219
- 优先选择简单的模型, 138
- 有序回归, 262
- 余弦相似度, 227
- 召回率, 195, 209
- 整数型, 20
- 正定矩阵, 55
- 正规方程, 106, 153
- 正交矩阵, 55
- 正态分布, 43, 47

正则化项, 9, 111, 175, 178,

212

支持向量, 163

直方图, 89

指数分布, 43, 47

指数损失函数, 177, 265, 300,

301

秩, 55

中位数, 50, 84

中位数绝对偏差, 51

重复取样, 286

重取样, 185

重新加权迭代最小二乘法,

153

逐点排序, 10, 254, 262, 264

逐对排序, 10, 255, 265

逐列排序, 10, 255, 265

主成分分析, 62, 85

主特征向量, 59

主特征值, 59

属性, 74

注释语句, 14, 15

柱状图, 92

准确率, 193

子取样, 278, 316

字符型, 20

自变量, 3, 104

自然语言处理, 253

最小二乘法, 105, 214



机器学习是一个热门而又高深的话题。多年来，符号学习、统计学习、深度学习等一系列高高在上的名词使得大众对机器学习敬而远之，大数据时代，机器学习的广泛成功应用再次引爆了大众对机器学习的关注。

机器学习能够解决什么样的问题？如何使用机器学习解决实际问题？应该怎样选择算法？本书从实践出发回答这些问题。书中首先通过实际应用场景引出机器学习中的几类典型问题，然后着重介绍解决各类问题的实用算法，并利用R语言和相关的软件包来引导读者实际使用这些算法。

- 不懂机器学习？没有关系。本书不仅介绍了机器学习的基本概念和算法原理，还提供了完整的程序代码，助读者轻松上手、快速入门。
- 数学基础不够？没有关系。本书一方面突出对概念和原理的理解，尽可能淡化了对数学背景的要求；另一方面也介绍了必备的数学知识，便于读者查阅。
- 不会使用R语言？没有关系。本书介绍了R语言的基本知识及常用R软件包，两位作者更是亲手绘制了全书90%以上的插图，手把手教读者用R语言分析数据和展现结果。

掌握本书介绍的算法和对应的R软件包后，读者可以顺利地针对新问题、新数据选择和使用机器学习算法，在实践中获得更大收获。



叶杰平 滴滴研究院副院长、密歇根大学终身教授

这本书不厚，但却覆盖了用机器学习技术解决实际问题的主要步骤和常用算法。考虑到实践中大家更关注的是如何选择和使用算法，两位作者还使用R语言软件包来引导读者实际操作。与市面上对机器学习作一般性介绍的书籍相比，本书介绍的算法稍稍复杂一些，但也更加实用，书中讨论的内容正是实际应用机器学习解决问题时所需要掌握的内容。对于广大业界爱好者和相关专业研究生来说，这是一本理想的入门读物和参考书，因此我非常乐意向大家推荐本书。

陈震中 国家青年千人计划专家、武汉大学教授

这是一本非常贴近实际应用的机器学习著作。两位作者根据多年的一线科研和工程实践经验，选取了最典型的一些机器学习算法，既通俗易懂地介绍了原理，又给出了公开数据集上的R语言实践。行文风格方面，本书兼顾了高校师生和工程技术人员的需求，在理论与实践之间达成了一个较好的平衡，因此具有广泛的适用性，值得推荐。

闫胜业 南京信息工程大学教授

现有的机器学习书籍有些侧重于算法原理的讲解，对具体实现介绍得很少；有些侧重于基本概念和算法的实现，易于上手但难于把握算法原理的精妙细节之处。本书淡化了对数学背景知识的要求，突出了对常用算法的通俗讲解和基于R软件包的实现，便于读者快速上手，是一本不可多得的机器学习教材和自学参考书，“实用”二字实至名归。

李武军 南京大学副教授、博士生导师

本书从解决实际问题的角度介绍了五类常用的机器学习模型，包括回归模型、分类模型、推荐模型、排序模型和集成学习模型。作者不仅介绍了模型的基本原理，还介绍了特征工程、模型评价和选择等相关的知识。内容有深度但通俗易懂，有广度但不一味求全，具有很强的实用性。本书既适合机器学习初学者，也可以作为企业机器学习项目研发的参考书。

唐磊 Clari首席数据科学家

越来越多的岗位要求机器学习方面的专业知识。每年都有一些机器学习专业的应届毕业生加入我们的团队，但是我们发现很多人在如何应用机器学习实际问题方面还存在知识缺失的问题。这本书从要解决的问题类型出发，介绍了机器学习的各种基本概念以及那些最实用的算法，并全面阐述了使用机器学习解决问题的全过程，娓娓道来而又深入浅出，对于初学者来说是一本很好的入门读物，对于广大的机器学习从业者来说也是一本很好的参考书。尽管本人从事机器学习的研究和应用多年，但是阅读此书也感到受益匪浅。

戚晓光 微软高级数据科学家

很高兴孙亮博士和黄倩博士将他们在工业界多年应用机器学习积累的相关经验和成果整理出来，值得向大家强烈推荐这本书。在我们的工作实践中，所遇到的大部分问题不外乎回归、分类、推荐、排序诸类，而集成学习是我们在建模过程中使用最多的一类算法。本书系统地讲解了适用于这些问题的常用算法，并且介绍了R中相应的软件包。就实用性而言，是一本非常贴近实战的不可多得的好书。



异步社区 www.epubit.com.cn
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-44646-6



ISBN 978-7-115-44646-6

定价:79.00 元

分类建议：计算机/人工智能

人民邮电出版社网址：www.ptpress.com.cn